AndroVul: a repository for Android security vulnerabilities

Zakeya Namrud
zakeya.namrud.1@ens.etsmtl.ca
ETS Montreal
Montreal, Quebec, Canada

Sègla Kpodjedo
segla.kpodjedo@etsmtl.ca
ETS Montreal
Montreal, Quebec, Canada

Chamseddine Talhi
chamseddine.talhi@etsmtl.ca
ETS Montreal
Montreal, Quebec, Canada

ABSTRACT

Security issues in mobile apps are increasingly relevant as these software have become part of the daily life for billions. As the dominant OS, Android is a primary target for ill-intentioned programmers willing to exploit vulnerabilities and spread malwares. Significant research has been devoted to the identification of these malwares. The current paper aims to contribute to that research effort, with a focus on providing an additional benchmark of Android vulnerabilities to be used in the detection of malwares. Our proposal is AndroVul, a repository for Android security vulnerabilities, including dangerous permissions, security code smells and dangerous shell commands. Our work builds on AndroZoo, a well known Android app dataset, and proposes data on vulnerabilities for a representative sample of about 16,000 Android apps. Moreover, we briefly describe and make available the scripts we wrote to automate the extraction of security vulnerabilities, given a list of apps; this allows any researcher to readily recreate a custom repository build from his or her apps of interest. Finally, we propose preliminary findings on the effectiveness of the vulnerability metrics present in our dataset, with respect to the detection of malicious apps. Our results show that the collected metrics, as input to even basic classifiers, are enough to achieve competitive results with respect to some recent malware detection works. Overall, Androvul, with its scripts and datasets, is intended as a starting package for mobile security researchers interested in jump-starting their investigations.

CCS CONCEPTS

• **Security and privacy** → *Intrusion/anomaly detection and malware mitigation*; *Mobile platform security*;

KEYWORDS

Mobile security, static analysis, reverse engineering, mobile computing

ACM Reference Format:
Zakeya Namrud, Sègla Kpodjedo, and Chamseddine Talhi. 2019. AndroVul: a repository for Android security vulnerabilities. In *Proceedings of CASCON Conference (CASCON '19).* Toronto, ON, Canada, 8 pages.

1 INTRODUCTION

With a market share of about 75%[1], Android is without doubt the leading mobile Operating System (OS), and is well positioned to become the default OS for new applications centered on the Internet Of Things. Security aspects are thus increasingly relevant as there are more incentives for malware developers to target Android devices. Android security has been extensively researched and that effort has been helped by the AndroZoo[2] dataset put together by Allix et al[1] 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. We propose in this paper 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) scripts that will jumpstart their data collection. Our dataset includes data on 16,180 applications from Google Play store as well as third party stores[1]. Our scripts extracted from these apps' binaries (called APKs) 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 [7]. Furthermore, we propose in this paper, preliminary experiments aimed at probing the predictive power of these various sources of vulnerability. The rest of the paper is organized as follows. Section 2 and 3 propose some background and related work. Section 4 and Section 5 present our scripts and datasets. In Section 6, we propose a preliminary empirical study based on the datasets. Then, Section 7 proposes a short discussion and Section 8, the conclusion.

2 BACKGROUND

In this section, we briefly present the possible security vulnerabilities our dataset is focused on: dangerous permissions (as defined by the Android system), troubling code attributes (as collected by AndroBugs) and security code smells (as proposed in [7]).

2.1 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

[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/

access to system features such as the camera, Internet, personal contacts, SMS etc. Table 3 in the appendix proposes a list of the various dangerous permissions as defined by the official Android developer resource[5].

2.2 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[6], Warning[7], Notice[8] and Info[9].

2.3 Security Code Smells

"Code smells" refer to code source elements that may indicate deeper problems[2]. In [7], Ghafari et al. introduce 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[7] that they regrouped into five categories, such as Insufficient Attack Protection, Security Invalidation, Broken Access Control, Sensitive Data Exposure, and Lax Input Validation.

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. [1] 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.[8] 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. [12], 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 proposing empirical studies on large numbers of Android apps but few propose publicly available datasets. Among those, we can cite Munaiah et al.[13] which proposes data (app category, permissions etc.) 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.[9], which also proposes a dataset of security vulnerabilities but for open source systems (8,694). Similar to [12], we propose a subset of a well know mobile app repository; we start with AndroZoo while [12] builds on F-Droid. Similar to [12], we also propose scripts 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 scripts propose a unique offering for Android security researchers.

3.2 Classification

A primary use of our dataset will be as input for malware detection techniques. As announced in the introduction, we propose in this paper preliminary investigations into the usefulness of the vulnerability metrics present in our dataset when it comes to detect malwares. 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 works that are the most recent and the closest to our experiments.

Permissions, especially dangerous ones, have been used extensively as inputs to various classification approaches. A particular recent work of interest is Bhattacharya et al.[16], 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 4 groups and used as input for 15 different machine learning classifiers 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 works investigated features other than permissions for malware detection purposes. For instance, Sharma et al.[17], which tried to leverage Dalvik[11] opcode occurences for malware classification purposes. They selected 5531 android malwares from the DREBIN repository [22] 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 Sachdeva S et al.[18], which focused on features extracted through Mobile Security Framework, an open source tool dedicated to mobile app security[12]. The approach proposed in [18] 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.

[5]https://developer.android.com
[6]Confirmed vulnerability that should be solved (except for testing code)
[7]Possible vulnerability that should be checked by developers
[8]Low priority issue
[9]No security issue detected.
[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

4 ANDROVUL-T: THE TOOL

Our repository proposes scripts that allow, given a directory of APKs, the automatic generation of a CSV file with information on the apps vulnerabilities. To accommodate statistical analysis, each vulnerability corresponds to a column, to which we attach some quantitative data indicating its presence (dangerous permission), the certainty behind it (Androbugs vulnerability), or its weight (security code smell).

Figure 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.

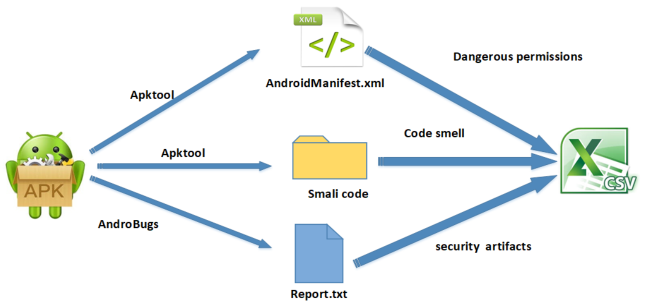


Figure 1: Overview of the AndroVul tool

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.

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.

As for the AndroBugs report, we parse it to extract vulnerabilities tagged Critical or Warning (see Section 2.2). To quantify the collected information for every vulnerability, we give a weight of 1 to Critical, 0.5 to Warning, and 0 otherwise.

We used regular expressions to parse the Smali code and extract security code smells defined in [7][14]. After which, we compute, for each vulnerability posed by a security code smell, a ratio indicating the relative presence of that vulnerability. We do so by dividing the number of identified instances of the code smell by the number of lines of codes in the Smali format.

For each of the extracted artifacts, Figure 2 illustrates the flow of our scripts.

5 ANDROVUL-D: THE DATASET

Androvul-D is our dataset of 73 vulnerability metrics collected on a sample of Android apps from AndroZoo. Figure 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 he wants in his 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 (category, etc.) about an app from its store.

[13]https://ibotpeaches.github.io/Apktool/

[14]Although that specific work did not expand much on the effectiveness of these regular expressions, a recently published follow-up work reported excellent detection accuracy.

5.1 Data Selection

For our data selection, we resorted to the AndroZoo dataset which contained, at the time we considered it[15], 5,848,157 apps. 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 Virus Total website (number of antiviruses that flag the app as a malware, scan date)[16]. Using a very simple sample size calculator[17], 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.

5.2 Dataset Structure

The dataset we propose consists in a simple CSV file containing information (as illustrated in Figure 4) about around 16K apps, 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, as described in Section 5.1
(2) The nine (9) code smells extracted from the reverse engineered Smali code
(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 73 metrics about dangerous permissions, Smali code smells and AndroBug-tagged vulnerabilities.

5.3 Dataset description

In this subsection, we provide some descriptive stats on our datasets, 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 within the last 3 years (2016-2018), 2 out of 3 within the last 5 years (2014-2018). The APK sizes range between 7 KB and 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%),

[15]May 2018

[16]More information here https://AndroZoo.uni.lu/lists

[17]https://www.surveysystem.com/sscalc.htm

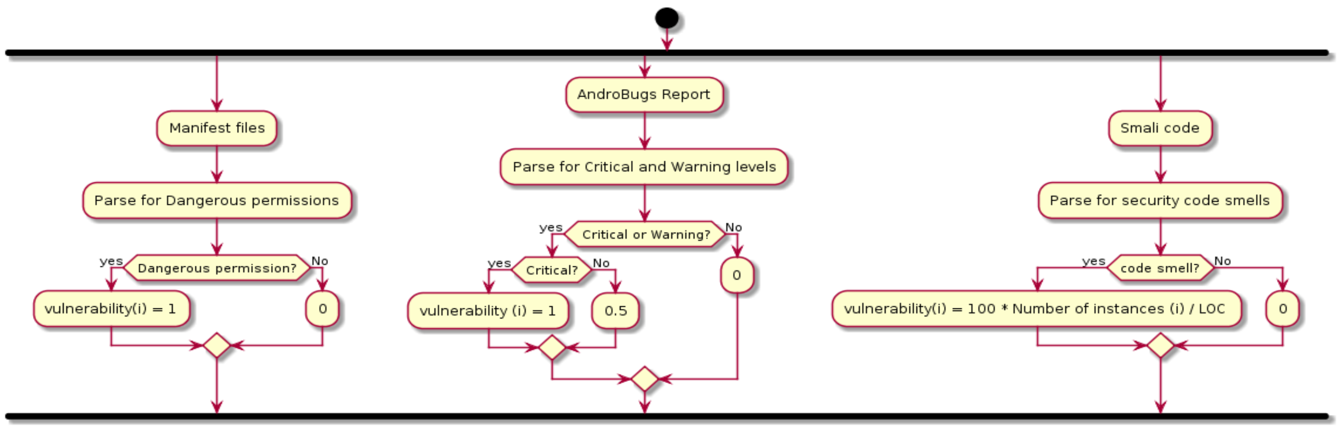


Figure 2: Parsing and Quantifying vulnerability data (i refers to the specific vulnerability)

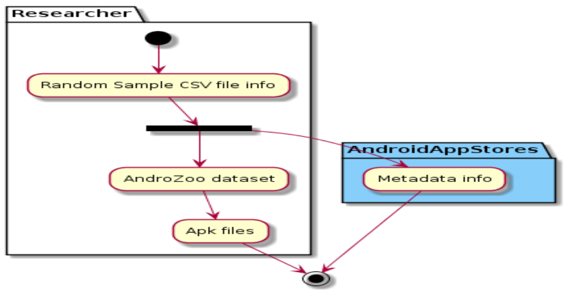


Figure 3: Data Selection and Gathering

appchina (10%), mi.com (2%) and anzhi (1%). When it comes to information from the antiviruses of TotalVirus[18], 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. Table presents some 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.

6 PRELIMINARY EMPIRICAL STUDY

In this section, we propose, as an example of the kind of uses that can be made of the AndroVul dataset, a preliminary investigation of said dataset when it comes to the detection of malwares. Androzoo does not explicitly tag apps as malwares. Rather, it provides the number of antivirus from TotalVirus that flag the app. For the purpose of this preliminary study, we will use, in accordance to the standard set by the DREBIN project [23] and used in other studies [19, 23–25], three (antivirus flags) as a threshold on which an app will be considered a malware.

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

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%)

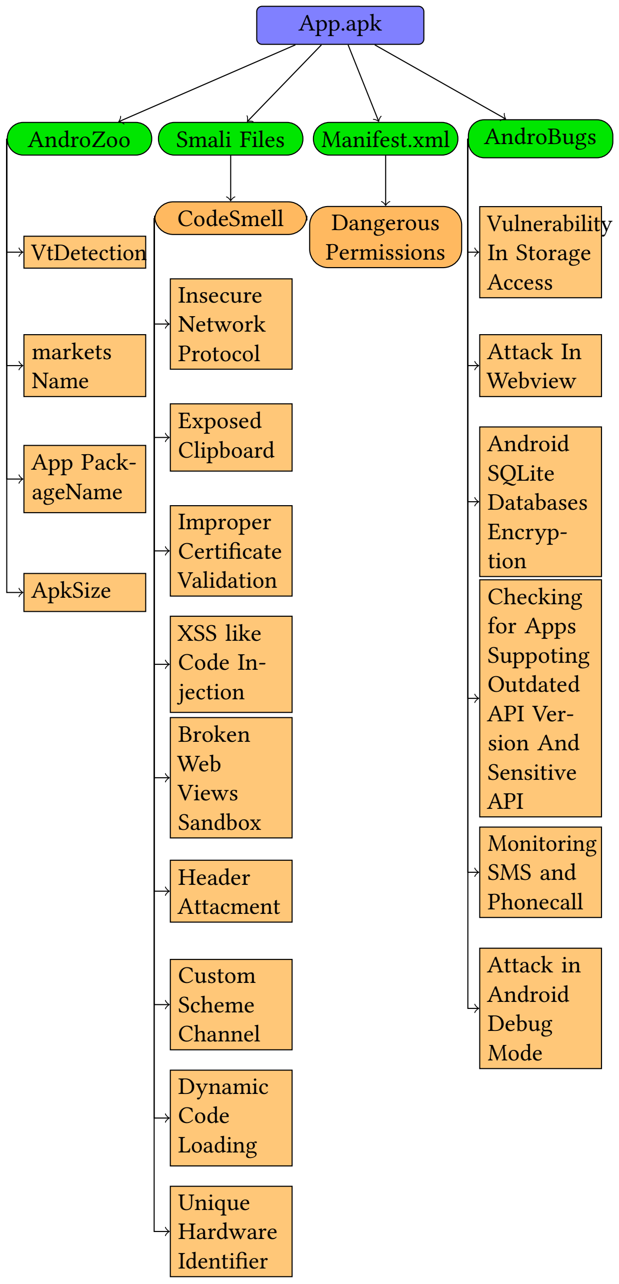
6.1 Datasets groups considered

To evaluate the dataset and the contribution of the 3 different categories to a classifier, we consider the following seven subsets:

(1) P: Permissions only
(2) S: Smell Code only
(3) A: AndroBugs only
(4) P+S : Permissions and Smell Code
(5) P+A : Permissions and AndroBugs
(6) S+A : Smell Code and AndroBugs
(7) All : Permissions and Smell Code and AndroBugs

6.2 Correlation Analysis

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



App.apk
AndroZoo
Smali Files
Manifest.xml
AndroBugs
CodeSmell
Dangerous Permissions
VtDetection
markets Name
App PackageName
ApkSize
Insecure Network Protocol
Exposed Clipboard
Improper Certificate Validation
XSS like Code Injection
Broken Web Views Sandbox
Header Attacment
Custom Scheme Channel
Dynamic Code Loading
Unique Hardware Identifier
Vulnerability In Storage Access
Attack In Webview
Android SQLite Databases Encryption
Checking for Apps Suppoting Outdated API Version And Sensitive API
Monitoring SMS and Phonecall
Attack in Android Debug Mode

Figure 4: Dataset Structure

flags and ii) a binary value representing the benign / malicious classification : 0 for benign or 1 for 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 the 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 way 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 Classifiers used

For this limited experiment, we selected four different classifiers representing 4 types of machine learning algorithms commonly used in the Android malware community. More precisely, we used the well known machine learning software Weka and selected Naive-Bayes (NB) from its *bayes* category, RBF classifier from its *function* category, JRip from its *rules* category, and J48 from its *tree* category. 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, 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 Performance Indicators

The application of classifiers result 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 papers 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 (AUROC)** 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 a perfect value.

6.5 Results

Figures 5 to 9 present the results obtained by the 4 classifiers on the seven dataset groups defined in Section 6.1.

Figures 5 and 6 respectively present data about precision and recall. When it comes to precision, the dataset *P+S* (Permissions +

Code Smell) gets the highest precision with nearly 0.83 for JRIP, with the dataset *All* being quite close at nearly 0.82. On the other end of the spectrum, the NB classifier with dataset *S* (Code Smell only) gets by far the worst precision with around 0.59. Unsurprisingly, the NB classifier with the Code Smell subgroup is also the combination that gets the best recall[19]. As observed for the precision values, the dataset including all metrics is also proposing mostly good recall values (around 0.85).

A careful look at the data suggests that Code Smells introduce greater variability in the classifiers performances, with datasets S (Code Smells) and S+A (Code Smells + AndroBugs), clearly the ones generating the starkest differences among the classifiers. This is in contrast with permissions, which seem to induce relatively close results for the classifiers: in particular for recall, results from different classifiers on datasets including permissions are remarkably close. Figure 7 displays F1-measures for all datasets and classifiers.

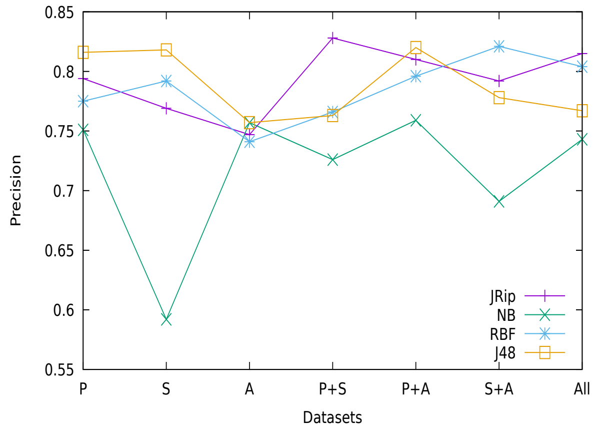


Figure 5: Precision over all datasets, with the 4 classifiers

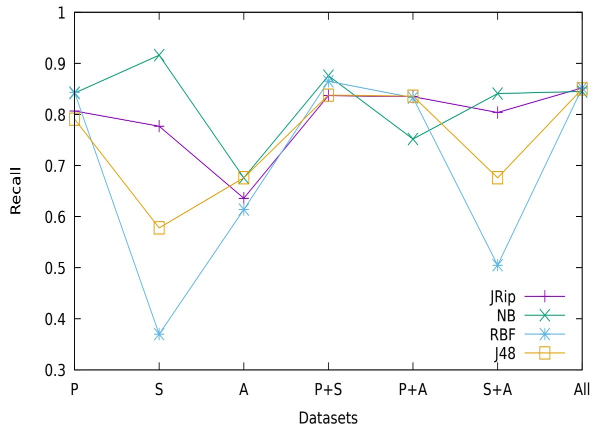


Figure 6: Recall over all datasets, with the 4 classifiers

Our observations about Code Smells seemingly introducing more variations in the results still hold. Not only that, dataset S (Code Smell only) gives some of the worst results, with the classifier RBF going as low as 0.5. We can also more clearly note that datasets that include Permissions not only are less dependent on which classifiers are used but additionnally tend to perform better than other variants. In particular, permissions by themselves (dataset P) yield a F1-measure around 0.8 while all metrics propose a F1-measure around 0.84. When it comes to the AUROC (Figure 8), the same

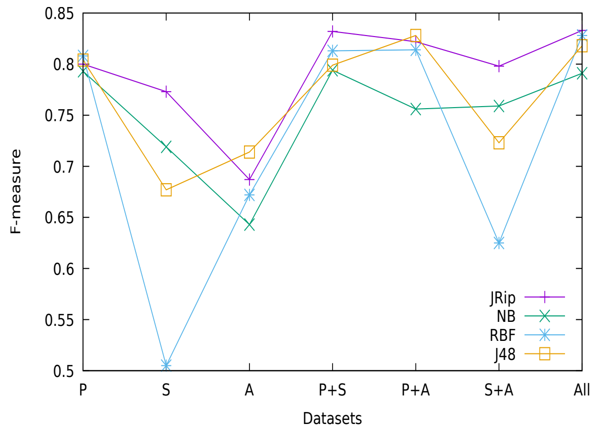


Figure 7: F1-Measure over all datasets, with the 4 classifiers

observations made for the F1-measure hold, only with generally higher values. Interestingly, for this performance measure, RBF is the classifier with both the worst result (0.66 with dataset 2) and the best (0.89 for dataset 7). Finally, as it relates to accuracy, which is a

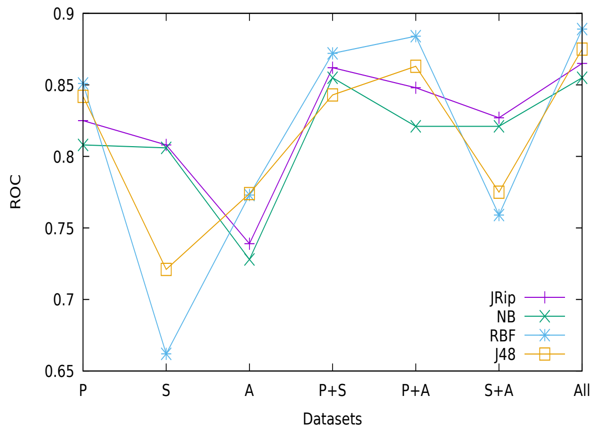


Figure 8: AUROC over all datasets, with the 4 classifiers

somewhat controversial performance measure[20], Figure 9 shows that there are no new insights, except possibly for slightly more variation on the classifiers performance. Overall, the dataset including all the metrics is the one with the consistently better results; with datasets P+S and P+A very close behind. As for classifiers,

[19]Recall often comes at the expense of precision, and vice versa.

[20]It can be misleading in some cases

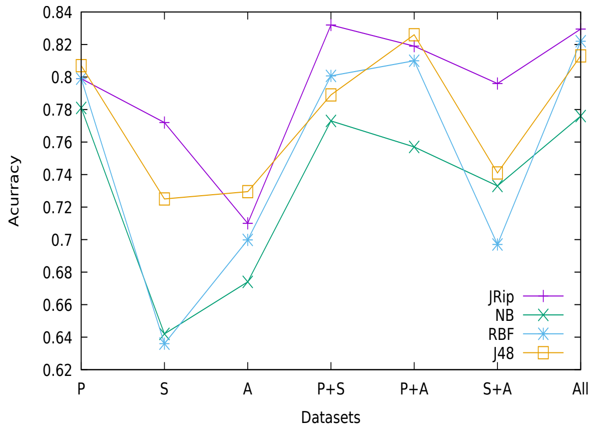


Figure 9: Accuracy over all datasets, with the 4 classifiers

there are no real winners across all datasets but when it comes to dataset All, JRIP and RBF seem to be the highest performers. In particular, we consider RBF on the dataset All to be the best combination from our experiments and provide in Table 2 a comparison between its results and some of the most notable related work. Table 2 shows that the extracted metrics can deliver even with barebone[21] results aligned with many recent related work. [18] does report a significantly better accuracy but it should be noted that the framework from which they extract their metrics provides data from dynamic analysis of the apps, which is a significant addition. Also, their experimental setup differs in some significant ways from ours, including a rather sophisticated and unique process to determine the ground truth of their classification, i.e., which apps are considered benign vs malicious. Nevertheless, these preliminary results are encouraging ones, with potential for improvement if we were to enhance the classifiers.

Table 2: Comparison with related work

Papers	Tool	Accuracy	F1	AUROC
Paper [16]	Weka	0.77	0.86	0.82
Paper [17]	Weka	0.79	?	?
Paper [18]	Weka	0.93	?	?
Our	Weka (RBF)	0.83	0.83	0.89

7 DISCUSSION: CHALLENGES, LIMITATIONS AND RESEARCH IDEAS

The construction of datasets such as the one proposed in this paper faces some challenges, big or small, including the automated downloading of all the selected APKs and the prior allocation of sufficient storage. Most of the effort resides in the coding of the scripts necessary to recover metadata of the app, set up the reverse engineering of the apk and execute the parsing of various artefacts. In particular, when it comes to the metadata, AndroZoo does not store that info so there is a need to retrieve it from the different stores. This is a tedious and sometimes unfruitful process since i) the apps may no longer be available in those stores, ii) the stores themselves do not offer simple ways to automatically get the info. Thus, HTML parsers have to be written for webpages that could be in other languages or have structures that can and do change. This is the main limitation of our dataset as many of the apps do not have complete metadata. To this effect, we plan some future work that would complement the data through the parsing of various app store clones that keep apps metadata even after they are deleted from the main market places. As it relates to our preliminary empirical study, some threats to validity are worth noting. An external threat to validity of our results is that we used only 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. Nevertheless, we cannot claim that our results would be generalizable. As for threats to internal validity, the threshold of three for the number of antivirus flags from which an app is considered malware is debatable but as previously said, it is a de facto consensus number among researchers working on Androzoo data. As such, using it, places our preliminary study in previously established grounds. Overall, we believe that the provided dataset can be used to test research ideas for anomaly detection or the mining of safe patterns useful for the identification of malwares. In addition, the accompanying scripts offer options for researchers wanting their own datasets. Following the simple instructions in the AndroVul GitHub repository, they can add their APKs (in the folder apks) and run the Python scripts.

[21]Machine learning classifiers generally require some additional customization and techniques (such as feature selection) to deliver their optimal results.

8 CONCLUSION

The ubiquity of smartphones, and their growing use make the security of these devices arguably as important as that of standard computers. In this paper, we propose a repository for Android vulnerabilities to better support the research community engaged with anomaly detection and security issues for Android apps. Our contributions are threefold. First, we propose scripts that harness well known reverse engineering tools and greatly simplify the generation of diverse vulnerability information (dangerous permissions, vulnerabilities from AndroBugs, and code smells in Smali code) for any app. Second, we propose vulnerability data on a random sample of 16,180 Android apps downloaded from the well established AndroZoo dataset. Our scripts and data made 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 own set of APKs. Third, we propose a preliminary empirical study that provide some insights into the predictive power of the vulnerability information mined by our scripts. Our scripts and data sample are available on GitHub and we intend to build and extend on that repository, notably by working on recovering more completely apps metadata.

REFERENCES

[1] Allix, K., Bissyandé, T.F., Klein, J. and Le Traon, Y., 2016, May. Androzoo: Collecting millions of android apps for the research community. In 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR) (pp. 468-471). IEEE.
[2] Shezan, F.H., Afroze, S.F. and Iqbal, A., 2017, January. Vulnerability detection in recent android apps: an empirical study. In 2017 International Conference on

Table 3: Dangerous Permissions (Definitions from Android)

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.

Networking, Systems and Security (NSysS) (pp. 55-63). IEEE.
[3] Wake, W.C., 2004. Refactoring workbook. Addison-Wesley Professional.
[4] Watanabe, T., Akiyama, M., Kanei, F., Shioji, E., Takata, Y., Sun, B., Ishi, Y., Shibahara, T., Yagi, T. and Mori, T., 2017, May. Understanding the origins of mobile app vulnerabilities: A large-scale measurement study of free and paid apps. In Proceedings of the 14th International Conference on Mining Software Repositories (pp. 14-24). IEEE Press.
[5] Palomba, F., Di Nucci, D., Panichella, A., Zaidman, A. and De Lucia, A., 2017, February. Lightweight detection of Android-specific code smells: The aDoctor project. In 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER) (pp. 487-491). IEEE.
[6] Hecht, G., Moha, N. and Rouvoy, R., 2016, May. An empirical study of the performance impacts of android code smells. In Proceedings of the International Conference on Mobile Software Engineering and Systems (pp. 59-69). ACM.
[7] Gadient, P., Nierstrasz, O. and Ghafari, M., 2017. Security in Android applications. PhD diss., Master s thesis. University of Bern.
[8] Geiger, F.X., Malavolta, I., Pascarella, L., Palomba, F., Di Nucci, D. and Bacchelli, A., 2018, May. 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). ACM.
[9] Gkortzis, A., Mitropoulos, D. and Spinellis, D., 2018, May. VulinOSS: a dataset of security vulnerabilities in open-source systems. In Proceedings of the 15th International Conference on Mining Software Repositories (pp. 18-21). ACM.
[10] Sarma, B.P., Li, N., Gates, C., Potharaju, R., Nita-Rotaru, C. and Molloy, I., 2012, June. Android permissions: a perspective combining risks and benefits. In Proceedings of the 17th ACM symposium on Access Control Models and Technologies (pp. 13-22). ACM.
[11] Viennot, N., Garcia, E. and Nieh, J., 2014, June. A measurement study of google play. In ACM SIGMETRICS Performance Evaluation Review (Vol. 42, No. 1, pp. 221-233). ACM.
[12] Krutz, D.E., Mirakhorli, M., Malachowsky, S.A., Ruiz, A., Peterson, J., Filipski, A. and Smith, J., 2015, May. A dataset of open-source Android applications. In Proceedings of the 12th Working Conference on Mining Software Repositories (pp. 522-525). IEEE Press.
[13] Munaiah, N., Klimkowsky, C., McRae, S., Blaine, A., Malachowsky, S.A., Perez, C. and Krutz, D.E., 2016, November. Darwin: A static analysis dataset of malicious and benign android apps. In Proceedings of the International Workshop on App Market Analytics (pp. 26-29). ACM.
[14] Ghafari, M., Gadient, P. and Nierstrasz, O., 2017, September. Security smells in Android. In 2017 IEEE 17Th international working conference on source code analysis and manipulation (SCAM) (pp. 121-130). IEEE.
[15] Gottschalk, M., Josefiok, M., Jelschen, J. and Winter, A., 2012. Removing energy code smells with reengineering services. INFORMATIK 2012.
[16] Bhattacharya, A. and Goswami, R.T., 2017. DMDAM: data mining based detection of android malware. In Proceedings of the First International Conference on Intelligent Computing and Communication (pp. 187-194). Springer, Singapore.
[17] Sharma, A. and Sahay, S.K., 2018. An investigation of the classifiers to detect android malicious apps. In Information and Communication Technology (pp. 207-217). Springer, Singapore.
[18] Sachdeva, S., Jolivot, R. and Choensawat, W., 2018. Android Malware Classification based on Mobile Security Framework. IAENG International Journal of Computer Science, 45(4).
[19] Wang, S., Chen, Z., Yan, Q., Yang, B., Peng, L. and Jia, Z., 2019. A mobile malware detection method using behavior features in network traffic. Journal of Network and Computer Applications, 133, pp.15-25.
[20] Pritam, N., Khari, M., Kumar, R., Jha, S., Priyadarshini, I., Abdel-Basset, M. and Long, H.V., 2019. Assessment of Code Smell for Predicting Class Change Proneness Using Machine Learning. IEEE Access, 7, pp.37414-37425.
[21] Stehman, S.V., 1997. Selecting and interpreting measures of thematic classification accuracy. Remote sensing of Environment, 62(1), pp.77-89.
[22] Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K. and Siemens, C.E.R.T., 2014, February. Drebin: Effective and explainable detection of android malware in your pocket. In Ndss (Vol. 14, pp. 23-26).
[23] Ali-Gombe, A., Ahmed, I., Richard III, G.G. and Roussev, V., 2016, March. Aspectdroid: Android app analysis system. In Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy (pp. 145-147). ACM.
[24] Li, W., Ge, J. and Dai, G., 2015, November. Detecting malware for android platform: An svm-based approach. In 2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing (pp. 464-469). IEEE.
[25] Li, J., Sun, L., Yan, Q., Li, Z., Srisa-an, W. and Ye, H.,2018.Significant permission identification for machine-learning-based android malware detection. IEEE Transactions on Industrial Informatics, 14(7), pp.3216-3225.
[26] Taylor,V.F.and Martinovic,I.,2017.A longitudinal study of financial apps in the Google Play Store.
[27] Darvish,H.and Husain, M.,2018,December.Security analysis of mobile money applications on android.In 2018 IEEE International Conference on Big Data(Big Data)(pp. 3072-3078). IEEE.