Overview of machine learning methods for Android malware identification

[1]Joao Lopes,˜ [1]Carlos Serrao,˜ [1,2]Lu´ıs Nunes, [1,3]Ana Almeida, [1,2]Joao Oliveira˜
[1]ISTAR - Information Sciences and Technologies and Architecture Research Center
[2]IT - Instituto de Telecomunicac¸oes˜
[3]CISUC - Centre for Informatics and Systems
ISCTE - Instituto Universitario de Lisboa´
Lisboa, Portugal
Email: Joao Pedro Lapa, carlos.serrao, luis.nunes, ana.almeida, joao.p.oliveira{@iscte-iul.pt}

Abstract—Mobile malware is growing and affecting more and more mobile users around the world. Malicious developers and organisations are disguising their malware payloads on apparently benign applications and pushing them to large app stores, such as Google Play Store, and from there to final users. App stores are currently losing the battle against malicious applications proliferation and existing malware. Detection methods based on signatures, such as those of an antivirus, are limited, new approaches based on machine learning start to be explored to surpass the limitations of traditional mobile malware detection methods, analysing not only static characteristics of the app but also its behaviour. This paper contains an overview of the existing machine learning mobile malware detection approaches based on static, dynamic and hybrid analysis, presenting the advantages and limitations of each, and a comparison between the reviewed methods.

Index Terms—security, malware, mobile, android, machine learning

I. Introduction

Mobile computing has achieved a level that has never been seen before (estimates are that the number of smartphones will reach 6,1 billion by 2020) [1]. The two major mobile platforms (Android and iOS) completely dominate the market and users continue to adhere massively to these mobile platforms. Users are switching from more traditional data processing platforms (such as desktop computers) and increasingly using mobile platforms for tasks such as messaging, e-commerce, productivity tools, health and fitness, home banking or payments [2]– [4]. Some of these are applications that handle very sensitive user's data.

As the trust of end-users in these mobile platforms and applications increases, more and more use them on a daily basis. However, as the number of users increases, as well as the amount of critical information deployed on these mobile platforms they become more attractive to attackers that will try to obtain unauthorized access to mobile devices and users data.

These mobile platforms are increasingly targeted by attackers, both on iOS and Android [5]. Android, due to its market penetration and openness is more attractive to attackers. Android is free and open, it is currently the operating system of nearly 80% of all mobile devices in the World and smart-device manufacturers use it as the basis for their systems [6]. Although the attacks' nature is quite distinct, one of the most common ways of attacking Android users is through the distribution of malware that disguises as legitimate applications. Mobile malware is on the rise and researchers found that 87% of all Android smartphones are exposed to at least one critical vulnerability [7]. Multiple types of malware exist for Android and could affect the user in a multiplicity of ways: banking malware, mobile ransomware, mobile spyware, MMS malware, mobile adware, and SMS trojans.

Users can be tricked into installing malware-infected applications and this is a real menace to mobile platforms. Malware applications, disguised as legitimate applications installed on the users' mobile device can access different areas of the device, other applications, all sorts of stored data, and capture data in transit. Therefore, from a security perspective, it is of utmost importance to be able to detect, identify and prevent the proliferation of malware in the mobile distribution chain (this chain includes developers, applications distributors stores, and end users).

Given that the majority of mobile users download their applications from application stores (such as the Google Play Store on the Android platform and the App Store on the iOS platform), this overview may prove useful for application store owners in order to improve their store's security, thus decreasing the number existing malware, as well as decreasing the number of potential malicious applications that try to penetrate their application marketplaces in the future.

This paper starts by providing a short introduction to the mobile malware problems and the existing limitations on its identification and eradication. Next, the different mobile malware detection and identification methods based on machine learning approaches are presented, covering both static, dynamic and hybrid detection methods. In this section, the major advantages and disadvantages of each are presented. Finally, in the last part of this paper, some conclusions drawn from the analysis conducted are exhibited

and a comparison between the different mobile malware detection methods studied is presented.

II. Mobile malware detection and identification

One of the most important actions for malware prevention is to be able to accurately detect and identify it. It is important to be able to automate the detection and identification processes, using intelligent methods, due to enormous amounts of applications being submitted to application stores and to mobile devices. This section presents an analysis of state of the art machine-learning based malware detection methods, based on static, dynamic and hybrid analysis of mobile applications.

A. Malware Detection Methods Based on Static Analysis Data

Static analysis consists on the analysis of a given application source code without executing it [8]. Particularly in Android, it implies the analysis of the contents of the Application Package (APK) file.

This type of analysis has the advantage of being fast and low on resource consumption. However, it it vulnerable to both code obfuscation techniques and dynamically loaded code [9] [10].

Sanz et al. [11] developed a static malware detection method that leverages the contents of the AndroidManifest.xml file, which describes essential information about a given application to the Android build tools, Google Play, but most importantly, to the Android operating system [12]. In order to retrieve this file from the APK, it uses a tool named Android Asset Packaging Tool (AAPT) [13].

Two specific fields from this file were used as features: uses-permission, which lists every permission that the application needs to operate correctly and uses-feature, which declares hardware and software features the application needs (for instance, the compass sensor) [12].

The features mentioned before were used to train the following algorithms: Logistic Regression (LR), Nave Bayes (NB), Bayesian Network (BN), Sequential Minimal Optimization (SMO), an implementation of K-Nearest Neighbours (K-NN) named IBk, Decision Tree (J48), Random Tree (RT) and Random Forest (RF). To train these algorithms, it was used a dataset comprised of 249 malware samples and 357 benign samples.

Peiravian and Zhu [14] developed a malware detection framework that uses permissions and Application Programming Interface (API) calls as features. This information is obtained using the reverse engineering tool Apktool [15], which extracts the AndroidManifest.xml file as well as the class files from a given APK. For a given application, the permissions are extracted from the AndroidManifest.xml file and are embedded in a binary vector P, where $P_i = 1$ if the *ith* permission is requested in its AndroidManifest.xml file, otherwise $P_i = 0$. The API calls are extracted from the class files, following the procedure explained above. As a result, every application is represented by a single binary vector of permissions and API calls in addition to a benign or malicious class label.

These features were used to train the following algorithms: Support Vector Machine (SVM), Decision Tree (DT) and Bagging [16]. To train these algorithms, it was used a dataset comprised of 610 malware samples and 1250 benign samples.

TABLE I
Performance of Malware Detection Methods Based on Static Analysis Data

References	Features	AUC
[11]	Permissions + Used Features	0.890 (LR)
		0.780 (NB)
		0.790 (BN)
		0.860 (SMO)
		0.900 (IBK)
		0.860 (J48)
		0.850 (RT)
		0.920 (RF)
[14]	Permissions	0.917 (J48)
		0.920 (SVM)
		0.956 (Bagging)
	API Calls	0.918 (J48)
		0.957 (SVM)
		0.956 (Bagging)
	Permissions + API Calls	0.936 (J48)
		0.963 (SVM)
		0.991 (Bagging)

Almin and Chatterjee [17] developed a permission-based malware detection method through an application that is composed of five major components.

The first and second components consist on identifying a users installed applications and extracting the permissions, respectively. The former is done using the Android PackageManager class using the getPackageManager method and the latter using the PackageInfo class, which holds all the information that is present on an AndroidManifest.xml file, such as permissions. The third component trains a clustering algorithm, namely KNN, using a vector of the permissions of a given application as input. This process's objective is twofold: creating clusters that represent different malware families as well as creating a benign applications cluster. The fourth component trains the NB algorithm in order to classify applications more accurately, since the previous step could have produced false positives. In order to train this algorithm, both the cluster number as well as the set of permission combinations that occur in a cluster are used as features. The last component presents a user with the list of malicious applications that were detected as well as the option to delete them.

Table I displays the performance of the studied Android malware detection methods based on static analysis, using Area Under the Curve (AUC) as the performance metric.

Observing table I, it can be concluded that the algorithm that achieves the best performance is Bagging. Furthermore, the combined use of permissions and API calls achieves better results than using them separately.

B. Malware Detection Methods Based on Dynamic Analysis Data

Contrary to static analysis, dynamic analysis consists of the execution of a given application in a sandboxed environment, in order to monitor its behaviour. This type of analysis has the advantage of being able to detect unknown malware although it demands more computational power and is more timeconsuming than static analysis [9].

Singh and Hofmann [18] developed a malware detection method that uses the frequency of system calls as features. The first stage of this process consists of executing each application of the sample set, which is comprised of 216 malicious samples and 278 benign samples, in an emulator using a tool named Monkey [19]. This tool generates pseudorandom user actions (clicks, touches, gestures and system-level events) [20]. Afterwards, a total of 337 Linux system calls of each application are monitored, resulting in a feature vector of 337 elements, where each element represents how many times that specific system call was invoked during runtime. In the next stage, every system call that has zero variance is removed from the feature vector, resulting in a final feature vector of 43 attributes, excluding the class label. This feature vector is used to train the following algorithms: DT, RF, Gradient Boosted Trees (GBT), KNN, SVM, Artificial Neural Networks (ANN) and Deep Learning (DL). In order to improve the performance of the chosen classifiers, three feature weighing techniques were also applied before training and testing the algorithms once more, namely, Information Gain (IG), Chi-square statistic and correlation.

Bhatia and Kaushal [21] developed an Android malware detection solution that also uses the frequency of invoked system calls at runtime as features. Using a dataset comprised of 50 malicious samples and 50 benign samples, every application is executed in an Android Virtual Machine (VM) using the Monkey tool for one minute, generating 500 gestures with a 500 millisecond delay between each event, while the Linux command strace is executed in parallel to extract the frequencies of every invoked system call during that period. This information is aggregated in a single matrix where each row represents the frequency of the system calls of a given application and each column represents the frequency of a given system call for every application. Afterwards, this matrix is converted to a .csv file that is used to train two algorithms: J48 and RF.

Afonso, de Amorim, Grgio, Junquera, and de Geus [22] developed a malware detection system leveraging the frequency of both API and system calls that are invoked at runtime.

In order to extract the API calls, the tool APIMonitor [23] is executed for five minutes while it is being executed on an emulator using the tool MonkeyRunner [24], which generates random events automatically (such as sending keystrokes). Furthermore, the file that handles the collection of API calls contained in this tool was modified in order to monitor additional API calls related to network access, process execution, string and file manipulation and information reading. The Linux command strace is also used during this period in order to extract the system calls. This information is aggregated into a vector of 74 API calls and 90 system calls, amounting to a total of 164 dimensions, each one representing how many times that particular API or system call was invoked. Using a dataset of 2295 malicious samples and 1485 benign samples,

TABLE II
PERFORMANCE OF MALWARE DETECTION METHODS BASED ON DYNAMIC ANALYSIS DATA

Reference	Features	F-score
[18]	System Calls, no feature weighing	0.946 (RF)
		0.943 (SVM)
		0.973 (DT)
		0.976 (GBT)
		0.901 (KNN)
		0.912 (ANN)
		0.937 (DL)
	System Calls, using IG	0.939 (RF)
		0.966 (SVM)
		0.972 (DT)
		0.981 (GBT)
		0.961 (KNN)
		0.952 (ANN)
		0.977 (DL)
	System Calls, using Chi-square statistic	0.946 (RF)
		0.966 (SVM)
		0.967 (DT)
		0.981 (GBT)
		0.960 (KNN)
		0.946 (ANN)
		0.965 (DL)
	System Calls, using correlation	0.961 (RF)
		0.969 (SVM)
		0.972 (DT)
		0.991 (GBT)
		0.986 (KNN)
		0.920 (ANN)
		0.968 (DL)
[21]	System Calls	0.850 (J48)
		0.885 (RF)
[22]	System Calls and API Calls	0.968 (RF)

the following algorithms were trained in order to determine which one will be used by the proposed method: RF, J48, LR, NB, BN, SMO, and IBk. RF achieved the best performance using the dataset mentioned above, so it was tested afterwards using a dataset comprised of 2257 malware samples and 1483 benign samples.

In order to evaluate the comparative performance of the analysed papers, we can use the Precision and Recall values to compute their respective F-scores, since not every dataset is balanced regarding their benign sample to malicious sample ratio. The F-score of an algorithm is given by the following equation (Equation 1):

$$Fscore = 2 \times \frac{(Precision \times Recall)}{(Precision + Recall)} \quad (1)$$

Table II displays the performance of the studied Android malware detection methods based on dynamic analysis, using F-score as performance metric.

Observing table II, the best performing algorithm is GBT using correlation as the feature weighing algorithm.

C. Malware Detection Methods Based On Hybrid Analysis

Hybrid analysis consists on combining both static and dynamic analysis in order to overcome their respective limitations with the main purpose of achieving better detection results [10].

Zhao, Xu and Zhang [25] developed a system that extracts permissions and API calls as static features and runtime behaviour as dynamic features in order to classify applications. In the static analysis process, a tool named Androguard [26] is used to extract the permissions from the AndroidManifest.xml file, resulting in a permission feature set that is further optimized in order to remove features that rarely appear. This results in a binary permission feature vector of 45 dimensions, representing the presence of each permission in a given application. Additionally, the API calls of applications from various sample sets are extracted through the analysis of their respective classes.dex files, using both Androguard and a reverse-engineering tool named baksmali [27]. In order to optimize the obtained API feature vector, the filter feature selection algorithm named Relief [28] is used, resulting in a final API call feature set of 22 dimensions where each dimension represents an API call. In the dynamic analysis process, every application is installed and executed on an emulator. In order to extract runtime behaviours as features, the tool Monkey [19] is executed while another called DroidBox [29] monitors the runtime behaviour to determine whether a given application exhibits malicious behaviour such as automatic network connection, malicious SMS sending, private information logging, among others. Additionally, the number of occurrences of each behaviour is registered and the Relief algorithm is used to remove irrelevant features, resulting in a final feature vector of 20 dimensions such as battery usage, user activity, network features, among others. Afterwards, this information is aggregated into a single feature vector with 87 dimensions. Using a dataset comprised of 359 malware samples and 500 benign samples, 150 malicious samples and 150 benign samples were chosen randomly to form both training and testing datasets, which were used by the following algorithms: SVM, KNN, NB, DT and RF.

Liu, Zhang, Li and Chen [30] developed a method that employs static analysis or dynamic analysis depending on the result of the APK extraction process. Using the tool Apktool [15], if it can successfully decompile a given application, it proceeds to the static analysis stage. However, if it does not produce useful information (for instance, if code obfuscation techniques were used) it employs dynamic analysis.

In the static analysis stage, the AndroidManifest.xml file is extracted from each application and every permission is mapped to a feature vector of 151 dimensions, each dimension representing a single permission. Additionally, every API call is extracted using the tool baksmali and is mapped to a feature vector of 3262 dimensions, each dimension representing an API call. Both feature vectors are joined afterwards, resulting in a final feature vector of 3413 dimensions. In the dynamic analysis phase, a system call feature vector of 345 dimensions is created where each dimension represents the frequency of the invoked system calls. To extract these features, the ADB (Android Debug Bridge) tool [20] is used. Afterwards, the application is executed and the invoked system calls are monitored using the Linux command strace while the Monkey operates it. Using a dataset comprised of 500 malicious samples and 500 benign samples, the following algorithms were trained: KNN, SVM and NB.

Arshad et al. [31] developed a hybrid malware detection model where the static analysis phase is carried out on a remote server and the dynamic analysis phase is employed on the device. This model is composed of two major components: a client application in which the dynamic analysis process occurs, and a remote server that handles the static analysis process as well as the training and testing of the machine learning algorithms.The client application is developed in order to let the user dynamically analyse an application of his/her choice. Once it does, the client application hooks the strace command with that application which monitors its invoked system calls. The client application was programmed to monitor the frequency of 10 specific system calls related to file operations and network access were called. Afterwards, a system call log file is generated and sent to the remote server. In the static analysis phase, the remote server receives the application identifier through the client application, and the server queries its database in order to check if it was not previously classified. If so, a report is sent back to the user, otherwise the server downloads the application and employs static analysis. This process consists of the extraction of several features such as requested hardware components, requested permissions, application components (services, broadcast receivers and content providers), Intent filters, suspicious API calls and restricted API calls. The first four are extracted from the application's AndroidManifest.xml file using the AAPT (Google, 2019a) tool. The last two are extracted from disassembling the application code from the classes.dex file using the baksmali tool. Afterwards, the remote server generates both static and dynamic binary feature vectors in order to train the following algorithms: SVM, RF, DT and NB. The mentioned algorithms were evaluated using the Drebin dataset [32], which is comprised of 5560 malware samples and 123453 benign samples.

Table III displays the performance of the studied Android malware detection methods based on hybrid analysis, using Accuracy as performance metric.

Observing table III, and given that [25] and [30] use balanced datasets, their results are comparable. Therefore, it can be

concluded that the algorithm that achieved the best performance is SVM using both permissions and API calls as features.

III. Conclusions

As the number of users using smartphones and mobile applications continues to grow also the security risks to which users' data is exposed are also increasing. These mobile platforms are already a target of choice for attackers/criminals that are exploring several attacks to compromise the users. One of the biggest risk mobile application users are exposed to is the impersonation of mobile applications applications that advertise a given set of functionalities but underneath operate in an obscure manner as a way to compromise users data, and eventually launch additional attacks against the users (sniffing data, extracting private information, redirecting users to malicious sites, steal payment information, downloading extra functionalities or other malicious applications, or simply displaying obnoxious and intrusive advertising). This type of applications is commonly designated as malware. Malware is a serious problem in mobile platforms, in particular, if they manage to be present on the official application stores to be download by the smartphone users - every other month, Google Play Store removes hundreds of malicious applications from the store.

TABLE III
Performance of Malware Detection Methods Based On Hybrid Analysis Data

References	Features	Accuracy (%)
[25]	Static	85.74 (NB)
		88.19 (J48)
		92.07 (RF)
		91.27 (SVM)
		84.56 (KNN)
	Hybrid	84.52 (NB)
		89.34 (J48)
		94.89 (RF)
		93.66 (SVM)
		86.71 (KNN)
[30]	Permissions	93.33 (NB)
		96.52 (SVM)
		95.58 (KNN)
	API Calls	94.23 (NB)
		99.07 (SVM)
		98.42 (KNN)
	Permissions + API Calls	94.41 (NB)
		99.28 (SVM)
		98.66 (KNN)
	System Calls	90.00 (NB)
		85.75 (SVM)
		87.92 (KNN)
[31]	Static	91.60 (NB)
		99.07 (RF)
		98.97 (SVM)
		98.56 (DT)
	Dynamic	62.07 (NB)
		82.76 (RF)
		82.76 (SVM)
		72.41 (DT)

Due to the enormous amount of mobile applications that are submitted to stores (tens of thousands per day), it is important to automate the detection of malicious applications [33]. Traditional anti-virus tools based on signatures have limited effectiveness on malware detection because of malware authors are constantly innovating the way to develop their applications. Therefore, more intelligent and adaptive ways are required to effectively detect evolutionary mobile malware. There are different strategies for malware detection based on machine learning: static, dynamic and hybrid analysis. The state-of-the-art analysis conducted on the existing machine learning techniques to help detect mobile malware, revealed that the use of machine learning techniques, in general, achieve high performance when applied to Android malware detection. This means that the usage of machine learning techniques will play a major role in the development of future Android malware detection solutions, in order to control the rampant proliferation of malware in the Android platform.

Static analysis-based malware detection methods proved to be a simple and fast way to detect malware. However, they are highly vulnerable to code obfuscation techniques, meaning that they might only be effective as a preliminary line of defence against known malware. The main types of features used in these methods are permissions and API calls, and their combined use always achieves better performance than the use of each of them separately. Dynamic analysis-based malware detection methods achieved high performance as well but are more time and resource consuming than their static counterparts. However, dynamic analysis methods are more effective on the detection of new malware as well as variations of existing malware. This type of method could be used in order to detect the appearance of new malware in a deliberate manner. Hybrid analysis-based malware detection methods use both static and dynamic analysis in tandem in order to cover their individual weaknesses, therefore being more versatile. However, only one of the analysed solutions combined both static and dynamic features in order to train machine learning algorithms and while achieving good accuracy, it also showed very high false positive rates which is not ideal. This type of method seems to be the one that has most potential, given that it is the most exhaustive one.

Another important aspect to consider in the development of future machine learning-based malware detection solutions are the used datasets. Every existing approach considers only small datasets, which can lead to poor generalisation on the trained models, since they rely heavily on the data that they observe during the training phase. Moreover, the datasets need to be varied, meaning that they need to represent a large amount of the malware landscape. Finally, the datasets need to be balanced regarding the number of malware and benign samples, especially when the metric used to measure an algorithm's performance is its accuracy.

ACKNOWLEDGEMENTS

This work is part of the AppSentinel project, co-funded by Lisboa2020/Portugal2020/EU in the context of the Portuguese Sistema de Incentivos a I&DT - Projetos em Copromoc¸ ao˜ (project 33953).

REFERENCES

[1] Ericsson, “Ericsson Mobility Report”, Tech. Rep., Ericsson, 2018.

[2] Dave Chaffey, “Mobile Marketing Statistics compilation”, 2018.

[3] Liran Einav, Jonathan Levin, Igor Popov, and Neel Sundaresan, “Growth, Adoption, and Use of Mobile E-Commerce”, *American Economic Review*, vol. 104, no. 5, pp. 489–494, 2014.

[4] Jaeki Song, Jeff Baker, Ying Wang, Hyoung Yong Choi, and Anol Bhattacherjee, “Platform adoption by mobile application developers: A multimethodological approach”, *Decision Support Systems*, vol. 107, pp. 26–39, 2018.

[5] Parvez Faruki, Ammar Bharmal, Vijay Laxmi, Vijay Ganmoor, Manoj Singh Gaur, Mauro Conti, and Muttukrishnan Rajarajan, “Android security: A survey of issues, malware penetration, and defenses”, *IEEE Communications Surveys and Tutorials*, vol. 17, no. 2, pp. 998– 1022, 2015.

[6] IDC, “Smarphone Market Share”, 2019.

[7] Daniel R Thomas, Alastair R Beresford, and Andrew Rice, “Security metrics for the android ecosystem”, in *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*. ACM, 2015, pp. 87–98.

[8] Belal Amro, “Malware Detection Techniques for Mobile Devices”, *International Journal of Mobile Network Communications & Telematics (IJMNCT)*, vol. 76, no. 45, pp. 1–10, 2017.

[9] Ms Prajakta, D Sawle, and A B Gadicha, “Analysis of Malware Detection Techniques in Android”, *International Journal of Computer Science and Mobile Computing*, vol. 33, no. 3, pp. 176–182, 2014.

[10] Saba Arshad, Munam A. Shah, Abdul Wahid, Amjad Mehmood, Houbing Song, and Hongnian Yu, “SAMADroid: A Novel 3-Level Hybrid Malware Detection Model for Android Operating System”, *IEEE Access*, vol. 6, pp. 4321–4339, 2018.

[11] Borja Sanz, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero, Pablo Garcia Bringas, and Gonzalo Alvarez, “PUMA: Permission usage´ to detect malware in android”, *Advances in Intelligent Systems and Computing*, vol. 189 AISC, pp. 289–298, 2013.

[12] Google, “Android Debug Bridge (adb) — Android Developers”, 2019. [13] Google, “AAPT2 — Android Developers”, 2019.

[14] Naser Peiravian and Xingquan Zhu, “Machine learning for Android malware detection using permission and API calls”, *Proceedings International Conference on Tools with Artificial Intelligence, ICTAI*, pp. 300–305, 2013.

[15] ApkTool, “Apktool - A tool for reverse engineering 3rd party, closed, binary Android apps.”, 2019.

[16] Leo Breiman, “Bagging predictors: Technical Report No. 421”, *Machine Learning*, , no. 2, pp. 19, 1994.

[17] Shaikh Bushra Almin and Madhumita Chatterjee, “A novel approach to detect Android malware”, *Procedia Computer Science*, vol. 45, no. C, pp. 407–417, 2015.

[18] Latika Singh and Markus Hofmann, “Dynamic Behavior Analysis of Android Applications for Malware Detection”, *2017 International Conference on Intelligent Communication and Computational Techniques (ICCT)*, , no. 2013, pp. 1–7, 2017.

[19] Google, “UI/Application Exerciser Monkey — Android Developers”, 2019.

[20] Google, “App Manifest Overview — Android Developers”, 2019.

[21] Taniya Bhatia and Rishabh Kaushal, “Malware detection in android based on dynamic analysis”, *2017 International Conference on Cyber Security And Protection Of Digital Services, Cyber Security 2017*, 2017.

[22] Vitor Monte Afonso, Matheus Favero de Amorim, Andr Ricardo Abed Gregio, Glauco Barroso Junquera, and Paulo Lcio de Geus, “Identifying´ Android malware using dynamically obtained features”, *Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 1, pp. 9–17, 2015.

[23] “droidbox/APIMonitor at master · pjlantz/droidbox · GitHub”.

[24] “monkeyrunner — Android Developers”, 2019.

[25] Yang Zhao, Guangquan Xu, and Yao Zhang, *Quality, Reliability, Security and Robustness in Heterogeneous Systems*, vol. 234, Springer International Publishing, 2018.

[26] Anthony Desnos and Subho Halder, “Androguard”, 2019.

[27] Ben Gruber, “BakSmali”, 2019.

[28] Girish Chandrashekar and Ferat Sahin, “A survey on feature selection methods”, *Computers and Electrical Engineering*, vol. 40, no. 1, pp. 16–28, 2014.

[29] Patrik Lantz, “Droidbox”, 2017.

[30] Yu Liu, Yichi Zhang, Haibin Li, and Xu Chen, “A hybrid malware detecting scheme for mobile Android applications”, *2016 IEEE International Conference on Consumer Electronics, ICCE 2016*, pp. 155–156, 2016.

[31] Saba Arshad, Munam A. Shah, Abdul Wahid, Amjad Mehmood, Houbing Song, and Hongnian Yu, “SAMADroid: A Novel 3-Level Hybrid Malware Detection Model for Android Operating System”, *IEEE Access*, vol. 6, pp. 4321–4339, 2018.

[32] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and¨ Konrad Rieck, “Drebin: Effective and Explainable Detection of Android Malware in Your Pocket”, *Proceedings 2014 Network and Distributed System Security Symposium*, , no. August, 2014.

[33] Ana Fernandes, Antnio Ravara, and Joo Casal, “App Threat Analysis: Combining static analysis with users feedback to accelerate app store resposne to mobile threats”, Tech. Rep., Universidade Nova de Lisboa, 2018.