


<b>ITC 2/48</b> Journal of Information Technology and Control Vol. 48 / No. 2 / 2019 pp. 235-249 DOI 10.5755/j01.itc.48.2.21457	<b>What Static Analysis Can Utmost Offer for          Android Malware Detection</b>	
	Received 2018/08/16	Accepted after revision 2019/03/28
	 <a href="http://dx.doi.org/10.5755/j01.itc.48.2.21457">http://dx.doi.org/10.5755/j01.itc.48.2.21457</a>	

# What Static Analysis Can Utmost Offer for Android Malware Detection

**Abdullah Talha Kabakus**

Duzce University, Faculty of Engineering, Department of Computer Engineering, Duzce, 81620, Turkey,  
 e-mail: talhakabakus@gmail.com

Corresponding author: talhakabakus@gmail.com

Malicious applications are widespread for Android despite the taken serious actions by the operating system. Static and dynamic analysis techniques are utilized to detect malware by identifying the signatures of malicious applications by inspecting both the resources and behaviors of malware, respectively. In this study, what static analysis can utmost offer to detect malware in Android ecosystem is discussed and experimented on commonly used datasets in the literature by proposing a novel Android malware detection approach based on static analysis techniques. With the proposed study, the effectiveness of novel static analysis features' in terms of detecting malware in Android ecosystem are proved. These features were underestimated by the related work in the literature. The experimental result shows that the proposed Android malware detection approach is very effective in terms of detecting Android malware. Each feature used by the proposed approach is evaluated by using different types of machine learning techniques in order to highlight its impact on detecting malware and inform the digital investigators. The accuracy of the proposed static analysis approach is calculated as high as 0.987 for 10,865 applications.

**KEYWORDS:** Android malware, Android malware detection, static analysis, machine learning, Android.

## 1. Introduction

Android is an open source mobile operating system which is owned by Google and powered by the contribution of eighty-four technology and mobile companies under a group named *Open Handset Alliance (OHA)*<sup>1</sup>. During Google I/O 2017, Google has an-

nounced that there are more than 2 billion monthly active Android devices in use around the world [15, 59]. According to a recent report by *Statista*, with being used by 87.7% of smartphones, Android has dominated the global smartphone market in the second quarter of 2017 [73]. The reasons behind this domination can be listed as follows: (1) Being an open source

<sup>1</sup> <https://www.openhandsetalliance.com>

operating system which makes it possible to install and customize for free [85], (2) ability to extend the default features of operating system by installing a large number of applications which are available in the official application market (namely *Play Store*<sup>2</sup>) [46], (3) being powered by a group of eighty-four companies which contain some widely used hardware manufacturers such as *Samsung, LG, Sony*. Unfortunately, this huge popularity attracts the attention of malicious application developers. According to the security researchers at *Check Point*, about 2 million Android users are affected from a malware dubbed as ‘*FalseGuide*’ which hides its malicious action in over forty fake companion guide applications for popular mobile games such as *Pokemon Go* and *FIFA Mobile* [47]. The Android malware ‘*Judy*’ is thought to reach as many as 36.5 million users on Play Store [53, 78]. Android malware aims (1) privilege escalation, (2) turning the infected devices into bots for remote control, (3) causing financial charges to infected users, and (4) sensitive information collection [7, 29, 35, 40, 45, 57, 62, 69, 76, 81, 82, 84-87, 90]. Despite this popularity and the level of threat, many researchers report the lack of security awareness of Android users [9, 27, 33, 38, 41, 45, 54, 72, 85]. Android application developers are required to explicitly declare the permissions that the developed application needs to demand from users through the use of provided classes and methods of Android API (Application Programming Interface). The practice recommended by Android official documentation while developing Android applications is minimizing the number of permissions that the application requires which are defined in application manifest file (*AndroidManifest.xml*) [68]. Since when a necessary permission is not defined in *AndroidManifest.xml* the application crashes, developers tend to demand more permissions than the application actually needs [30]. For this reason, some tools such as *PScout* [79] and *Androguard*<sup>3</sup> are proposed. Despite the existence of these tools, Android application developers cannot solely rely on these tools since (1) Android API is being updated regularly, and (2) these tools are based on program analysis [10]. Also, it is reported that some research-

ers find the official Android documentation is incomplete which leads applications to be overprivileged [30, 79]. The Android security mechanism is solely based on permissions which are needed to be granted by the users in order to let Android applications access sensitive contents such as contacts, messages or hardware such as camera, telephony [11, 24, 30, 4, 51]. Despite its importance, a report that measures the awareness and interest level of Android users for the permissions mechanism shows that while 42% of participants are even unaware of the existence of permissions, only 17% of participants pay attention to permissions during the installation process [31]. *Google Play Protect* is announced during Google I/O 2017 which is an always-on service bundled with the Play Store application and scans the installed applications on device regularly in order to ensure that the applications remain benign over the time [4, 21]. According to Android Security Center, *Google Play Protect* checks applications, settings and critical security data from over 2 billion Android devices and over than 50 billion applications are verified per day by comparing application behavior across these devices thanks to the used machine learning techniques [67].

Android malware detection approaches are generally divided into two categories: (1) Static analysis, and (2) dynamic analysis. Static analysis approaches use reverse engineering techniques to obtain application source files in order to detect malicious applications without executing applications [3, 27, 32, 34, 90]. The other one, dynamic analysis approaches execute applications in a controlled and instrumented environment such as a sandbox or a virtual machine in order to monitor their runtime behavior such as network access, memory modifications, and track dynamic taint [76]. The main objective of this study is revealing what can static analysis utmost offer for Android malware detection with the use of resources of applications. For this reason, a novel static analysis approach is proposed which introduces some novel features. The proposed approach combines these static analysis features with various machine learning techniques. The proposed approach is evaluated on large and commonly used datasets in order to make a conclusion about the effectiveness of the used static analysis features. The rest of the paper is structured as follows: Section 2 presents the related work.

<sup>2</sup> <https://play.google.com/store>

<sup>3</sup> <https://github.com/androguard/androguard>

Section 3 describes the proposed static analysis technique with explaining how each feature is extracted. Section 4 discusses the findings and the experimental result. Finally, Section 5 concludes the paper with future directions.

---

## 2. Related Work

In this section, Android malware detection approaches are briefly reviewed. Android malware detection approaches are generally classified through the technique they use: (1) Static analysis techniques, and (2) dynamic analysis techniques.

### 2.1. Static Analysis

*Kirin* [27] is a security service that evaluates an application's demanded permissions and checks them against a set of security rules to mitigate malware at installation time by modifying the Android Application Installer. *Kirin* solely relies on the permissions defined in *AndroidManifest.xml* file rather than examining whether these permissions are actually used by the application. *SCanDroid* [32] extracts the information from the Android manifest file and application source code which are used to decide if the application may lead to unwanted information flows. *Stowaway* [30] uses static analysis techniques in order to detect the overprivilege by mapping the set of API calls that an application uses with the related permissions. *Stowaway* is evaluated with a set of 940 applications and the experimental result shows that one-third of these applications are overprivileged. *DroidAPIMiner* [1] conducts a frequency analysis to capture the most relevant API calls and utilizes top malware used APIs as features. Then these features are used within *KNN (K-Nearest Neighbors)* algorithm to classify applications as malicious or benign. *Drebin* [6] uses static analysis to gather the characteristics of Android applications. Then *Drebin* utilizes *SVM (Support Vector Machines)* to classify applications as malicious or benign. *Apex* [55] makes dynamic permission revocation possible which lets the user revoke granted permissions when the application is installed. *APK Auditor* [39] is a permission-based static analysis system which extracts the permissions of application defined on the Android manifest file and calculates a novel malware score for

each application based on the usage of permissions. The calculated score is compared to the malware threshold limit which is determined by using logistic regression on the database that stores previously analyzed both malicious and benign applications. Then *APK Auditor* classifies an application as malicious if the calculated score exceeds the malware threshold limit. *SAMADroid* [8] proposes Android malware detection model based on the three different levels such as (1) static and dynamic analysis, (2) local and the remote host, and (3) machine learning intelligence. The static analysis features that *SAMADroid* are based on both the *AndroidManifest.xml* file (e.g., requested hardware components, requested permissions, and application components) and the detected API calls. *Sayfullani et al.* [66] presented a static algorithm based on the extraction of resources from the *.apk* file of applications, namely, (1) *AndroidManifest.xml*, (2) *classes.dex* which contains the compiled source files in *.dex* format, and (3) *resources.arsc* which contains the compiled resources. They propose *Normalized Bernoulli Naive Bayes* classifier which is an improved *Naive Bayes* classifier that resulted in higher accuracy according to the experimental result. *Bao et al.* [10] proposed two static analysis approaches which are (1) the approach that utilizes a collaborative filtering technique inspired by the intuition that applications that have similar features usually demand similar permissions, (2) the approach recommends permissions thanks to a technique that utilizes *Naive Bayes multinomial classification* algorithm to build a prediction model by analyzing the descriptions of applications which are available on Play Store. The limitations of this approach are (1) it detects API usages by the existence of related import statements but not every declared import statement is used by source code which is also known as unused import statement, and (2) it only considers classes from Android SDK and Java standard libraries but developers define their own classes thanks to the inheritance mechanism provided by Java programming language. *APPSPEAR* [48] is an automated Android unpacking system for both *Dalvik* and *ART (Android Runtime)* Android application runtime environments that are proposed to overcome code protections which are commonly used by malware to hide their malicious aims as *Symantec* reports that the ratio of packed malware has increased to 25% by August 2016 [70]. *DroidDet* [91]

is a static analysis tool that utilizes *Rotation Forest* as the machine learning algorithm based on the static analysis features such as permissions, system events, and the rate of sensitive APIs.

## 2.2. Dynamic Analysis

*Crowdroid* [14] traces Linux system calls, converts them into feature vector in order to use as the features of utilized *K-means* clustering algorithm. *MADAM* (*a Multi-level Anomaly Detector for Android Malware*) [25] is a dynamic analysis tool that combines features at the kernel-level and at the application level and utilizes machine learning techniques to perform malware analysis. The major drawback of *MADAM* is that it performs monitoring and analyzing processes on the device which is not applicable since mobile devices generally have limited computation (e.g., CPU) and storage capabilities (e.g., memory, disk, battery) [25, 50, 77]. Some dynamic analysis approaches [13, 37, 44] utilize power consumption as the main feature for malware detection. This approach surely is useful to detect malware which is specifically designed to consume the battery of device but this approach is very limited when the large varieties of Android malware are considered [3, 88]. *TaintDroid* [26] is dynamic taint-tracking and analysis system that simultaneously tracks sensitive data such as location, microphone, and camera. They report that 15 applications of randomly selected 30 popular applications have reported locations of users to a remote server for advertising. In addition to that, approximately one-third of the applications have exposed some information about the phone which is specific to the device. *Paranoid Android* [60] transfers the execution trace recorded by a tracer located in the smartphone to a server located in the cloud which replays the execution trace within the replica of the mobile phone. Canfora et al. [17] proposed an Android malware detection approach based on sequences of system calls. They use machine learning techniques in order to associate sequences of system calls with malicious behaviors. The biggest advantage of this approach is that it is able to cope with the dynamism of the mobile application ecosystem which is commonly underestimated by the related work since it can detect unknown malware. *SAMADroid* [8] analyzes the traces generated by system calls during the execution of

an Android application. *DroidTrace* [89] is a *ptrace*<sup>4</sup> based dynamic analysis system with forward execution capability which utilizes the *ptrace* to monitor calls of a process in order to classify the payloads behaviors through the system calls.

## 3. Material and Method

In this section, the detail of the proposed approach and the dataset which is used to evaluate it are described in the following subsections.

### 3.1. Data Collection Process

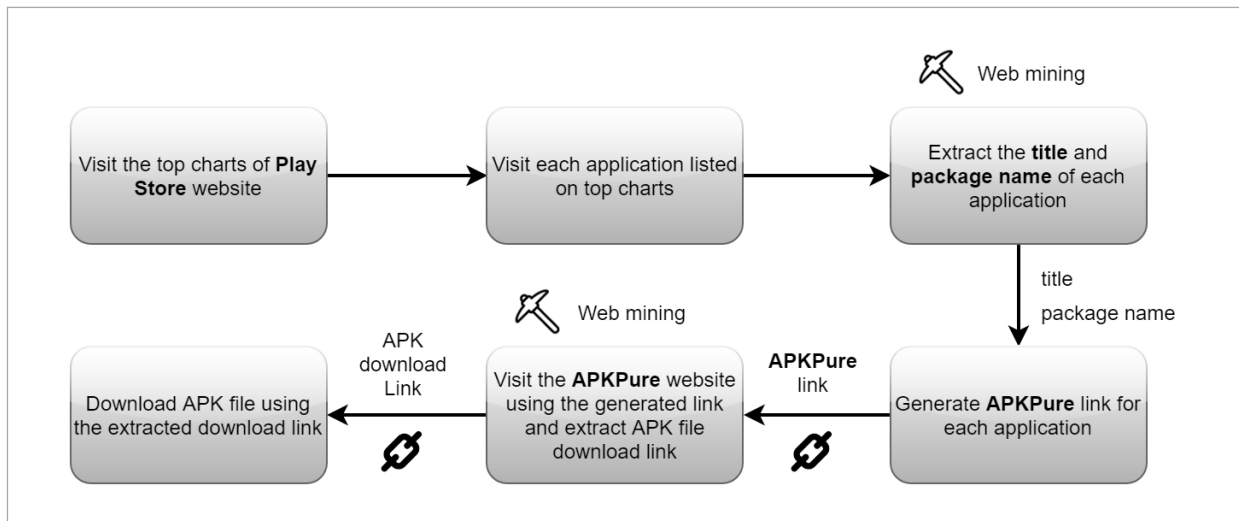
The approach proposed in [38] is used to fetch applications from Play Store which constructs the benign application dataset of the proposed approach. Applications stored in Play Store are downloaded by the usage of a website named *APKPure* which provides an introduction webpage that contains information about each application. Alongside that, this webpage also provides a link to download the *apk* (Android Package) file of the application which is an archive file that contains whole resources of the application. The applications that construct the benign dataset of the proposed approach are selected from top charts in order to decrease the probability of being malicious [17] and belong to different categories (e.g., games, education, business, family, communication, medical) in order to reflect the variety of applications in Android ecosystem. Applications' titles are retrieved from the website of Play Store utilizing web mining techniques and the related package names are extracted from the Play Store URLs (Unified Resource Locator) of applications which can be retrieved from the path variable "*id*". Then the *APKPure* introduction webpage of the related application is retrieved programmatically using the following URL pattern: "[https://apkpure.com/application-title/package\\_name](https://apkpure.com/application-title/package_name)" with converting the title to lowercase format with replacing the spaces with "\_". Similar to the application's title, the download link of each application is extracted from *APKPure* utilizing web mining techniques since *APKPure* does not provide any APIs to retrieve the metadata about an Android application which is stored on its knowledge-base. The whole process of downloading applications from Play Store is presented in Fig. 1.

<sup>4</sup> *ptrace* is a Unix system call which enables one process to observe and control the execution of another process.



**Figure 1**

The whole process of downloading applications from Play Store



### 3.2. Dataset Information

Malicious applications are obtained from datasets which are widely used in the literature such as *Drebin* [6], *Android Genome Project*, and *F-Droid* [28]. To the best of my knowledge, *Drebin* is one of the biggest Android malware datasets available which contains malware that belong to 179 different malware families. As a total, the whole dataset used by the proposed static analysis approach contains 10,658 applications as the overview of the constructed dataset is listed in Table 1.

**Table 1**

The overview of the constructed dataset used by the proposed static analysis approach

Dataset	Type	Number of Samples
Applications collected from Play Store	Benign	2,902
<i>Drebin</i> [6]	Malicious	5,373
<i>Android Genome Project</i> [90]	Malicious	1,260
<i>F-Droid</i> [28]	Malicious	1,123

### 3.3. The Features of Proposed Static Analysis Approach

The features of the proposed static analysis approach are obtained from both (1) *AndroidManifest.xml* which is the file that contains declarations for the application's core components (e.g., activities, services, permissions), and (2) the application source code files (Java files) which are obtained by using reverse engineering techniques. The features extracted from the *AndroidManifest.xml* are (1) *the number of activities*, (2) *the number of services*, (3) *the number of receivers*, (4) *the number of features*, (5) *the number of dangerous permissions*, (6) *the number of custom permissions*, and (7) *the number of other permissions*. Unlike the other related static analysis approaches, the feature *number of permissions* are divided into three categories as (1) "dangerous" permissions which require users to grant them explicitly [63], (2) "custom" permissions which are defined by developers and not the ones Android operating system defines [23], and (3) "other" permissions indicate the permissions which do not belong to the first two categories. The only feature which is obtained from the decompiled Java source files is the *number of lines of code (loc)* of the application. To the best of our knowledge, this feature is novel since it has not been used by any related work. The features which are used by the proposed static analysis approach are listed in Table 2 with their sources and descriptions.

**Table 2**

The features which are used by the proposed static analysis approach with their sources and descriptions

Feature	Source	Description
Number of activities	<i>AndroidManifest.xml</i>	The number of activities defined on the <i>AndroidManifest.xml</i> file
Number of services	<i>AndroidManifest.xml</i>	The number of services defined on the <i>AndroidManifest.xml</i> file
Number of receivers	<i>AndroidManifest.xml</i>	The number of receivers defined on the <i>AndroidManifest.xml</i> file
Number of features	<i>AndroidManifest.xml</i>	The number of features defined on the <i>AndroidManifest.xml</i> file
Number of dangerous permissions	<i>AndroidManifest.xml</i>	The number of dangerous permissions defined on the <i>AndroidManifest.xml</i> file
Number of custom permissions	<i>AndroidManifest.xml</i>	The number of custom permissions defined on the <i>AndroidManifest.xml</i> file
Number of other permissions	<i>AndroidManifest.xml</i>	The number of other permissions defined on the <i>AndroidManifest.xml</i> file
Number of lines of code	Decompiled source code files	The total number of lines of code that decompiled Java source files contain

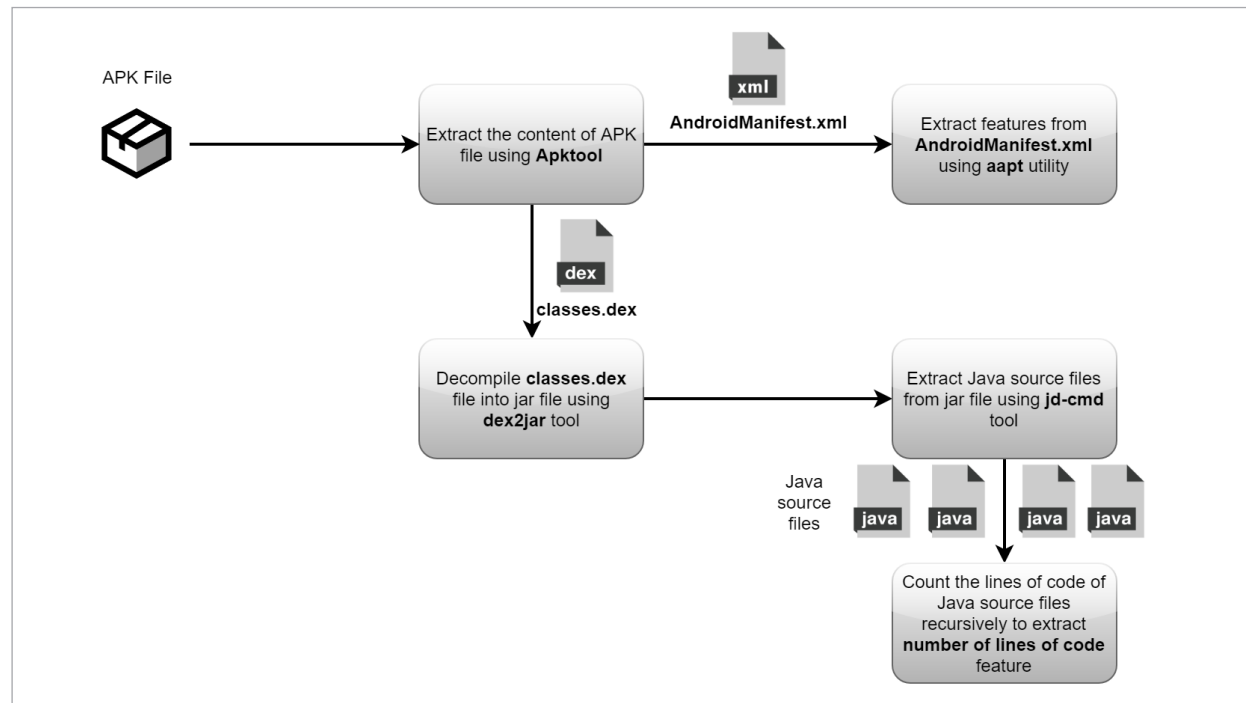
### 3.4. Extracting Features from APK File

The downloaded *apk* files are extracted using an open source third-party tool named *Apktool* [5]. *Apktool* extracts the *AndroidManifest.xml* and the compiled *dex* (*Dalvik executable*) file (*classes.dex*) from the

*apk* file. The *aapt* (*Android Asset Packaging Tool*) is a command line utility which comes bundled with Android SDK (Software Development Kit) [65]. A service, which is implemented using Java programming language, is used to (1) extract the *apk* file using

**Figure 2**

The whole process of extracting features of the proposed static analysis approach



*Apktool*, (2) read the contents of the extracted *AndroidManifest.xml* file using the *aapt* utility, and (3) extract the number of lines of code of the application. In order to extract the feature *number of lines of code* of the application, the implemented service decompiles the extracted *dex* file into the *jar* (Java archive) using an open source third-party tool named *dex2jar* [61]. Then the *jar* file is extracted into the Java source files using another open source third-party command line tool named *jd-cmd* [16]. Finally, the implemented service counts the number of lines of code of each extracted Java file recursively. The whole process of extracting features of the proposed static analysis approach is presented in Fig. 2.

## 4. Result and Discussion

The proposed static analysis approach was utilized with various machine learning algorithms in order to reveal which algorithm provides the best performance in terms of accuracy. Since the problem is a classification problem, the performance of the proposed system can be evaluated by using the confusion matrix. While positive means malicious applications, negative means benign applications. The terms *TP* (*True Positive*), *TN* (*True Negative*), *FP* (*False Positive*), and *FN* (*False Negative*) refer to the number of true positive instances, the number of true negative instances, the number of false positive instances, and the number of false negative instances, respectively. The performance of the proposed static analysis approach is evaluated by using five indexes namely *accuracy*, *precision*, *recall*, *F-measure*, and *MCC* (*Matthews Correlation Coefficient*) which are calculated as the following equations:

$$Accuracy = (TP + TN) / (TP + TN + FP + FN)$$

$$Precision = TP / (TP + FP)$$

$$Recall = TP / (TP + FN)$$

$$F - measure = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

$$MCC = \frac{(TP \times TN) - (FP \times FN)}{\sqrt{(TP + FP) \times (FN + TN) \times (FP + TN) \times (TP + FN)}}$$

*Precision* is the proportion of the number of correctly identified malware to the total number of malware.

*Recall* or *true-positive rate (TPR)* is the percentage of malware correctly identified. *F-measure* is the harmonic mean of the *precision* and *recall*. *MCC* is a correlation coefficient between the observed and predicted binary classifications, and is a value between -1 and 1. Whilst a coefficient of 1 indicates a perfect prediction, -1 indicates an inverse prediction. A coefficient of 0 indicates an average random prediction [64].

### 4.1. Feature Selection

Feature selection is a key process of machine learning in order to increase the accuracy of the system's prediction. For this reason, the information gain of each feature is experimented. In order to experiment the information gain each used feature provides, *CorrelationAttributeEval* attribute evaluator, which evaluates the worth of an attribute by measuring the correlation between it and the class [12, 20, 42, 43, 56], is used with the *Ranker* search method. As the experimental results listed in Table 3 show the novel feature *number of lines of code* provides the best information gain. Also, the feature *number of other permissions* provides the worst information gain which proves the categorization of permissions is useful in order to effectively classify applications through the permissions they demand.

**Table 3**

Information gain of each feature used by the proposed static analysis approach

Feature	Information Gain
Number of lines of code	0.808
Number of activities	0.363
Number of services	0.326
Number of receivers	0.324
Number of dangerous permissions	0.224
Number of custom permissions	0.177
Number of features	0.166
Number of other permissions	0.061

### 4.2. Evaluation of Machine Learning Algorithms

Various machine learning (ML) algorithms are utilized in order to compare their performance in terms of detecting malware. For this reason, the proposed

static analysis approach is evaluated with ten different machine learning algorithms namely (1) *KNN*, (2) *BayesNet*, (3) *NaiveBayes*, (4) *Logistic Regression*, (5) *SVM*, (6) *J48*, (7) *RandomForest*, (8) *RandomTree*, (9) *Bootstrap Aggregation (Bagging)*, and (10) *AdaBoost*. *KNN* is configured according to the number of neighbors. For the experiment, the number of neighbors is set to 1, 5, and the algorithms are named *KNN1*, *KNN5*, respectively. *Random Forest* is a forest that consists of configurable number of decision trees [80]. For the experiment, the number of decision trees is set to 100, 1000, and the algorithms are named *RandomForest100*, *RandomForest1000*, respectively. *SMO (Sequential Minimal Optimization)* algorithm is an improved training algorithm for *SVM* [58, 71] which is provided by *Weka*<sup>5</sup>, an open source widely used data mining software. *SMO-npolykernel (SMO-normalizedpolykernel)* and *SMO-polykernel* are implementations of the *SMO* algorithm according to the related kernel functions which are provided by *Weka*. 10-fold cross-validation is employed for the evaluation of each algorithm.

Thus, 9,539 samples are used for training, while the remaining 1,065 samples are used for testing. In 10-fold cross-validation, the dataset is randomly partitioned into ten equal sized parts where a single part is used for testing and the remaining nine parts are used for training. Since this process is repeated ten times, the whole dataset is used for both training and testing with ensuring that all samples are used once for validation [52]. The metrics that are used for the evaluation of each algorithm are *precision*, *recall*, *F-measure*, and *MCC* because of they are commonly used evaluation metrics by the related work [2, 10, 19, 22, 36, 46, 52, 74, 80, 83, 85, 87-88]. As the evaluation result of the proposed static analysis approach when it is utilized with a wide range of algorithms is listed in Table 4, the *precision*, *recall*, and *F-measure* of the proposed static analysis approach are calculated as high as 0.987 when the system is utilized with the *RandomForest* algorithm and the number of decision trees is set to 1,000. The same configuration produces the highest *MCC* value (0.966) as well.

**Table 4**

Performance comparison of the proposed static analysis approach when it is evaluated with various machine learning algorithms

ML Algorithm	Precision	Recall	F-measure	MCC
<i>KNN1</i>	0.977	0.977	0.977	0.943
<i>KNN5</i>	0.977	0.977	0.977	0.942
<i>BayesNet</i>	0.974	0.974	0.974	0.934
<i>NaiveBayes</i>	0.969	0.968	0.968	0.921
<i>Logistic Regression</i>	0.983	0.983	0.983	0.957
<i>SMO-polykernel</i>	0.976	0.975	0.975	0.938
<i>SMO-npolykernel</i>	0.967	0.967	0.967	0.917
<i>J48</i>	0.982	0.982	0.982	0.953
<i>RandomForest100</i>	0.986	0.986	0.986	0.965
<i>RandomForest1000</i>	0.987	0.987	0.987	0.966
<i>RandomTree</i>	0.98	0.98	0.98	0.951
<i>Bagging</i>	0.984	0.984	0.984	0.961
<i>AdaBoost</i>	0.982	0.982	0.982	0.954

<sup>5</sup> <https://www.cs.waikato.ac.nz/ml/weka/>



**Table 5**

Comparison of the proposed work with the related work in terms of utilized analysis technique and used features

Related Work	Analysis Technique	Used Features
<i>SCanDroid</i> [32]	Static analysis	Android manifest file and decompiled application source code
<i>Stowaway</i> [30]	Static analysis	Android manifest file and decompiled application source code
<i>DroidAPIMiner</i> [1]	Static analysis	Decompiled application source code
<i>Drebin</i> [6]	Static analysis	Android manifest file and decompiled application source code
<i>APK Auditor</i> [39]	Static analysis	Android manifest file
<i>DroidDet</i> [91]	Static analysis	Android manifest file and decompiled application source code
<i>Crowdroid</i> [14]	Dynamic analysis	Monitored system calls
<i>MADAM</i> [25]	Dynamic analysis	Monitored system calls and system resources
[13], [37], [44]	Dynamic analysis	Monitored power consumption
<i>TaintDroid</i> [26]	Dynamic analysis	Tracked taints during program execution
<i>Paranoid Android</i> [60]	Dynamic analysis	Execution trace of the program
[17]	Dynamic analysis	Monitored system calls
<i>SAMADroid</i> [8]	Dynamic analysis	Traces generated by system calls
<i>DroidTrace</i> [89]	Dynamic analysis	Monitored process calls

When the Android malware is investigated, it has been noticed that Android malware vary by the way they target to harm the device. Whilst some malicious applications tend to harm end-users through the provided activities which let users interact with the application like a benign one (i.e. playing a video game, sending a message, etc.), some others tend to utilize the services and receivers which contain some powerful features that Android SDK provides to complete malicious actions in the background. The way that a malware completes its malicious action changes through the version of the Android operating system that is running on the victim's device as the security mechanism of Android operating system evaluates. The main security mechanism of the Android operating system that is applied to applications is permissions as it is discussed in Introduction. Hence, the permissions of each application are extracted and specifically categorized in a similar way Android operating system itself categorizes permissions. The proposed system's malware detection mechanism is specifically designed not to base on a signature database that consists of several signatures of malware since being able to detect zero-day malware.

There are many Android malware detection approaches based on static analysis techniques as some of these approaches are briefly described in Section 2. Since the dataset used to evaluate the proposed Android malware detection approach is unique, it is not possible to directly compare the performances of related work in terms of malware detection accuracy. Instead of that, a comparison of the proposed approach with the related work in terms of utilized analysis technique and used features are listed in Table 5.

## 5. Conclusion

Static analysis techniques use static resources which are available before the installation and execution of Android applications. Since these resources contain all the resources the applications use in order to perform their actions, techniques based on static analysis are very effective in terms of malware detection. In addition to that, static analysis is lightweight compared to dynamic analysis since there is no need to monitor the executions of applications which requires various monitoring approaches to track taints

and monitor system resources. In this study, what static analysis can utmost offer for Android malware detection is experimented with the proposed approach which extracts features from static resources and utilizes various machine learning algorithms to detect malicious applications in a large dataset of 10,658 Android applications which is combined from both widely-used datasets and a collected dataset of applications available in Play Store. According to the experimental result, the proposed approach's accuracy is calculated as high as 0.987. The contributions of this study can be listed as follows:

- *Effective malware detection.* The proposed novel static analysis approach, which utilizes different types of machine learning algorithms, is capable of identifying Android malware with high accuracy.
- *Zero-day malware detection.* Since the proposed approach does not detect malicious applications through a signature database that consists of several signatures of malicious applications, it is designed to detect zero-day malware.
- *Novel features.* The proposed system uses some novel features such as the number of lines of code.
- *Lightweight analysis.* Linear time analysis and learning techniques are applied for efficiency. The proposed approach is capable of detecting malicious applications in larger datasets in a reasonable time.
- *Explainable approach.* The features used to train the proposed approach are explained and evaluated in order to reveal the efficiency of each feature. The feature which is utilized by the proposed approach for the first time “*the number of lines of code*” provides the best information gain alongside the used features which proves its efficiency.
- *Expandable knowledge base.* The proposed system's knowledge base is expandable since it is capable of downloading Android applications from Play Store automatically.
- *Fully automated mechanism.* The proposed system is fully automated as the system accepts an *apk* file as the input and extracts all the features thanks to the developed pipeline architecture.

As a future work, the proposed approach may be enhanced by source code analysis in order to interpret the real intentions of API calls. Also, the proposed system's knowledge base may be updated by including newer malicious application datasets when they exist.

### Acknowledgments

The author would like to thank Computer Security Group – the University of Göttingen for sharing the Drebin dataset and Zhou Y., Jiang X. for sharing the dataset of Android Malware Genome Project.

### Declaration of Conflicting Interests

The author declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

### Funding

The author received no financial support for the research, authorship, and/or publication of this article.

## References

1. Aafer, Y., Du, W., Yin, H. DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android. In 9th International Conference on Security and Privacy in Communication Networks (SecureComm 2013), Sydney, Australia, 2013, 86-103. [https://doi.org/10.1007/978-3-319-04283-1\\_6](https://doi.org/10.1007/978-3-319-04283-1_6)
2. Afonso, V. M., de Amorim, M. F., Grégio, A. R. A., Junquera, G. B., de Geus, P. L. Identifying Android Malware Using Dynamically Obtained Features. *Journal of Computer Virology and Hacking Techniques*, 2015, 11(1), 9-17. <https://doi.org/10.1007/s11416-014-0226-7>
3. Alzaylaee, M. K., Yerima, S. Y., Sezer, S. Improving Dynamic Analysis of Android Apps Using Hybrid Test Input Generation. In IEEE International Conference on Cyber Security and Protection Of Digital Services (Cyber Security 2017), London, UK, 2017. <https://doi.org/10.1109/CyberSecPODS.2017.8074845>
4. Android - Google Play Protect, 2019. Retrieved January 1, 2019, from <https://www.android.com/play-protect/>
5. Apktool - A Tool for Reverse Engineering 3rd Party, Closed, Binary Android Apps, 2018. Retrieved April 1, 2018, from <https://ibotpeaches.github.io/Apktool/>

6. Arp, D., Spreitzenbarth, M., Malte, H., Gascon, H., Rieck, K. Drebin: Effective and Explainable Detection of Android Malware in Your Pocket. In Symposium on Network and Distributed System Security (NDSS), San Diego, California, USA, 2014, 23-26. <https://doi.org/10.14722/ndss.2014.23247>
7. Arshad, S., Ahmed, M., Shah, M. A., Khan, A. Android Malware Detection Protection: A Survey. International Journal of Advanced Computer Science and Applications (IJACSA), 2016, 7(2), 463-475. <https://doi.org/10.14569/IJACSA.2016.070262>
8. Arshad, S., Shah, M. A., Wahid, A., Mehmood, A., Song, H. SAMADroid: A Novel 3-Level Hybrid Malware Detection Model for Android Operating System. IEEE Access, 2018, 6, 4321-4339. <https://doi.org/10.1109/ACCESS.2018.2792941>
9. Backes, M., Gerling, S., Hammer, C., Maffei, M., Backes, M., Gerling, S., Hammer, C. AppGuard - Real-Time Policy Enforcement for Third-Party Applications, 2012. Retrieved January 1, 2019, from <http://sps.cs.uni-saarland.de/publications/monitor.pdf>
10. Bao, L., Lo, D., Xia, X., Li, S. Automated Android Application Permission Recommendation. Science China Information Sciences, 2017, 60(9), 1-17. <https://doi.org/10.1007/s11432-016-9072-3>
11. Bläsing, T., Batyuk, L., Schmidt, A. D., Camtepe, S. A., Albayrak, S. An Android Application Sandbox System for Suspicious Software Detection. In 5th IEEE International Conference on Malicious and Unwanted Software (Malware 2010), Nancy, France: IEEE, 2010, 55-62. <https://doi.org/10.1109/MALWARE.2010.5665792>
12. Boonjing, V., Pimchangthong, D. Data Mining for Customers' Positive Reaction to Advertising in Social Media. In Proceedings of the Federated Conference on Computer Science and Information Systems, Prague, Czech Republic, 2017, 11, 945-948. <https://doi.org/10.15439/2017F356>
13. Buennemeyer, T. K., Nelson, T. M., Clagett, L. M., Dunning, J. P., Marchany, R. C., Tront, J. G. Mobile Device Profiling and Intrusion Detection Using Smart Batteries. In Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008), Waikoloa, HI, USA, 2008, 1-10. <https://doi.org/10.1109/HICSS.2008.319>
14. Burguera, I., Zurutuza, U., Nadjm-Tehrani, S. Crowdroid: Behavior-Based Malware Detection System for Android. In Proceedings of the 1st ACM workshop on Security and Privacy in Smartphones and Mobile Devices - SPSM'11, Chicago, IL, USA, 2011, 15. <https://doi.org/10.1145/2046614.2046619>
15. Burke, D. Android: Celebrating a Big Milestone Together with You, 2017. Retrieved January 1, 2019, from <https://www.blog.google/products/android/2bn-milestone/>
16. Cacek, J. kuart/jd-cmd: Command Line Java Decompiler, 2018. Retrieved January 2, 2019, from <https://github.com/kwart/jd-cmd>
17. Canfora, G., Medvet, E., Mercaldo, F., Visaggio, C. A. Detecting Android Malware Using Sequences of System Calls. In Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile - De-Mobile 2015, Bergamo, Italy, 2015, 13-20. <https://doi.org/10.1145/2804345.2804349>
18. Chandramohan, M., Tan, H. B. K. Detection of Mobile Malware in the Wild. Computer, 2012, 45(9), 65-71. <https://doi.org/10.1109/MC.2012.36>
19. Chang, Y.-C., Wang, S.-D. The Concept of Attack Scenarios and its Applications in Android Malware Detection. In 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Sydney, Australia: IEEE, 2016, 1485-1492. <https://doi.org/10.1109/HPCC-SmartCity-DSS.2016.0211>
20. Cheung, K. Y., Tong, K. K., Lee, K. H., Leung, K. S. Classification of RNAs with Pseudoknots Using K-mer Occurrences Count as Attributes. In 13th IEEE International Conference on BioInformatics and BioEngineering (IEEE BIBE 2013), Chania, Greece, 2013. <https://doi.org/10.1109/BIBE.2013.6701575>
21. Cunningham, E. Keeping You Safe with Google Play Protect, 2017. Retrieved January 1, 2019, from <https://blog.google/products/android/google-play-protect/>
22. Dash, S. K., Suarez-Tangil, G., Khan, S., Tam, K., Ahmadi, M., Kinder, J., Cavallaro, L. DroidScribe: Classifying Android Malware Based on Runtime Behavior. In Proceedings - 2016 IEEE Symposium on Security and Privacy Workshops (SPW 2016), 2016, San Jose, CA, USA, 252-261. <https://doi.org/10.1109/SPW.2016.25>
23. Define a Custom App Permission, 2018. Retrieved April 1, 2018, from <https://developer.android.com/guide/topics/permissions/defining.html>
24. Di Cerbo, F., Girardello, A., Michahelles, F., Voronkova, S. Detection of Malicious Applications on Android OS. In IWCF'10 Proceedings of the 4th International Conference on Computational Forensics, Aoyama, Tokyo, Japan: Springer-Verlag, 2010, 138-149. [https://doi.org/10.1007/978-3-642-19376-7\\_12](https://doi.org/10.1007/978-3-642-19376-7_12)

25. Dini, G., Martinelli, F., Saracino, A., Sgandurra, D. MA-DAM: A Multi-Level Anomaly Detector for Android Malware. In Kotenko, I., Skormin, V. (Eds.), *Computer Network Security*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, 7531, 240-253. <https://doi.org/10.1007/978-3-642-33704-8>
26. Enck, W., Gilbert, P., Chun, B.-G., Cox, L. P., Jung, J., McDaniel, P., Sheth, A. N. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*, Vancouver, BC, Canada, 2010, 393-407.
27. Enck, W., Ongtang, M., McDaniel, P. On Lightweight Mobile Phone Application Certification. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS'09)*, Chicago, Illinois, USA, 2009, 235-245. <https://doi.org/10.1145/1653662.1653691>
28. F-Droid - Free and Open Source App Repository, 2018. Retrieved March 6, 2018, from <https://f-droid.org>
29. Fan, W., Sang, Y., Zhang, D., Sun, R., Liu, Y. DroidInjector: A process injection-based dynamic tracking system for runtime behaviors of Android applications. *Computers and Security*, 2017, 70, 224-237. <https://doi.org/10.1016/j.cose.2017.06.001>
30. Felt, A. P., Chin, E., Hanna, S., Song, D., Wagner, D. Android Permissions Demystified. In *Proceedings of the 18th ACM conference on Computer and Communications Security - CCS'11*, New York, New York, USA: ACM Press, 2011, 627-638. <https://doi.org/10.1145/2046707.2046779>
31. Felt, A. P., Ha, E., Egelman, S., Haney, A., Chin, E., Wagner, D. Android Permissions: User Attention, Comprehension, and Behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security - SOUPS'12*, Washington, DC, USA, 2012, 1-14. <https://doi.org/10.1145/2335356.2335360>
32. Fuchs, A. P., Chaudhuri, A., Foster, J. S. SCanDroid: Automated Security Certification of Android Applications. Technical Report, Department of Computer Science, University of Maryland Read, 2009. <https://doi.org/10.1.1.164.6899>
33. Gibler, C., Crussell, J., Erickson, J., Chen, H. AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale. In *TRUST'12 Proceedings of the 5th International Conference on Trust and Trustworthy Computing*, Vienna, Austria, 2012, 7344 LNCS, 291-307. [https://doi.org/10.1007/978-3-642-30921-2\\_17](https://doi.org/10.1007/978-3-642-30921-2_17)
34. Grace, M., Zhou, Y., Wang, Z., Jiang, X. Systematic Detection of Capability Leaks in Stock Android Smartphones. In *Proceedings of the 19th Network and Distributed System Security Symposium (NDSS 2012)*, San Diego, California, USA, 2012, 1-15.
35. Grace, M., Zhou, Y., Zhang, Q., Zou, S., Jiang, X. RiskRanger: Scalable and Accurate Zero-day Android Malware Detection. In *Proceedings of the 10th international Conference on Mobile Systems, Applications, and Services - MobiSys'12*, Low Wood Bay, Lake District, United Kingdom: ACM Press, 2012, 281-294. <https://doi.org/10.1145/2307636.2307663>
36. Guido, M., Ondricek, J., Grover, J., Wilburn, D., Nguyen, T., Hunt, A. Automated identification of installed malicious Android applications. *Digital Investigation*, 2013, 10, 96-104. <https://doi.org/10.1016/j.diin.2013.06.011>
37. Jacoby, G. A., Davis, N. J. Battery-Based Intrusion Detection. In *Global Telecommunications Conference 2004 (GLOBECOM'04)*, Dallas, Texas, USA: IEEE, 2004, 4, 2250-2255. <https://doi.org/10.1109/GLOBECOM.2004.1378409>
38. Kabakus, A. T., Dogru, I. A. An In-depth Analysis of Android Malware Using Hybrid Techniques. *Digital Investigation*, 2018, 24, 25-33. <https://doi.org/10.1016/j.diin.2018.01.001>
39. Kabakus, A. T., Dogru, I. A., Cetin, A. APK Auditor: Permission-Based Android Malware Detection System. *Digital Investigation*, 2015, 13, 1-14. <https://doi.org/10.1016/j.diin.2015.01.001>
40. Kang, H., Jang, J. W., Mohaisen, A., Kim, H. K. Detecting and Classifying Android Malware Using Static Analysis Along with Creator Information. *International Journal of Distributed Sensor Networks*, 2015, 1-9. <https://doi.org/10.1155/2015/479174>
41. Kelley, P. G., Consolvo, S., Cranor, L. F., Jung, J., Sadeh, N., Wetherall, D. A Conundrum of Permissions: Installing Applications on an Android Smartphone. In Blyth, J., Dietrich, S., Camp, L. J. (Eds.), *Financial Cryptography and Data Security*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, 7398, 68-79. <https://doi.org/10.1007/978-3-642-34638-5>
42. Khalaf, A., Humadi, A. M., Akeel, W., Hashim, A. S. Students' Success Prediction Based on Bayes Algorithms. *International Journal of Computer Applications*, 2017, 178(7), 6-12. <https://doi.org/10.5120/ijca2017915506>
43. Khammas, B. M., Monemi, A., Bassi, J. S., Ismail, I., Nor, S. M., Marsono, M. N. Feature Selection and Machine Learning Classification for Malware Detection.



- Jurnal Teknologi, 2015, 77(1), 243-250. <https://doi.org/10.11113/jt.v77.3558>
44. Kim, H., Smith, J., Shin, K. G. Detecting Energy-Greedy Anomalies and Mobile Malware Variants. In Proceedings of the 6th international conference on Mobile Systems, Applications, and Services (MobiSys'08), Breckenridge, CO, USA: ACM, 2008, 239-252. <https://doi.org/10.1145/1378600.1378627>
  45. King, J., Lampinen, A., Smolen, A. Privacy: Is There an App for That? In Proceedings of the Seventh Symposium on Usable Privacy and Security (SOUPS'11), New York, NY, USA: ACM Press, 2011, 1-20. <https://doi.org/10.1145/2078827.2078843>
  46. Kumar, A., Kuppusamy, K. S., Aghila, G. FAMOUS: Forensic Analysis of MOBILE Using Scoring of Application Permission. Future Generation Computer Systems, 2018, 83, 158-172. <https://doi.org/10.1016/j.future.2018.02.001>
  47. Kumar, M. Beware! New Android Malware Infected 2 Million Google Play Store Users, 2017. Retrieved January 1, 2019, from <http://thehackernews.com/2017/04/android-malware-playstore.html>
  48. Li, B., Zhang, Y., Li, J., Yang, W., Gu, D. APPSPEAR: Automating the Hidden-Code Extraction and Reassembling of Packed Android Malware. Journal of Systems and Software, 2018, 140, 3-16. <https://doi.org/10.1016/j.jss.2018.02.040>
  49. Liang, S., Du, X. Permission-Combination-Based Scheme for Android Mobile Malware Detection. In 2014 IEEE International Conference on Communications (ICC), Sydney, Australia: IEEE, 2014, 2301-2306. <https://doi.org/10.1109/ICC.2014.6883666>
  50. Liu, L., Yan, G., Zhang, X., Chen, S. Virusmeter: Preventing Your Cellphone from Spies. Recent Advances in Intrusion Detection, 2009, 244-264. [https://doi.org/10.1007/978-3-642-04342-0\\_13](https://doi.org/10.1007/978-3-642-04342-0_13)
  51. Mahmood, R., Esfahani, N., Kacem, T., Mirzaei, N., Malek, S., Stavrou, A. A Whitebox Approach for Automated Security Testing of Android Applications on the Cloud. In 7th International Workshop on Automation of Software Test (AST 2012), Zurich, Switzerland: IEEE Press, 2012, 22-28. <https://doi.org/10.1109/IWAST.2012.6228986>
  52. Milosevic, N., Dehghantanha, A., Choo, K. K. R. Machine Learning Aided Android Malware Classification. Computers and Electrical Engineering, 2017, 61, 266-274. <https://doi.org/10.1016/j.compeleceng.2017.02.013>
  53. Morris, D. Z. Android Malware Judy' Hits as Many as 36.5 Million Phones, 2017. Retrieved January 1, 2019, from <http://fortune.com/2017/05/28/android-malware-judy/>
  54. Mylonas, A., Kastania, A., Gritzalis, D. Delegate the Smartphone User? Security Awareness in Smartphone Platforms. Computers and Security, 2013, 34, 47-66. <https://doi.org/10.1016/j.cose.2012.11.004>
  55. Nauman, M., Khan, S., Zhang, X. Apex: Extending Android Permission Model and Enforcement with User-Defined Runtime Constraints. In Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security (ASIA CCS'10), Beijing, China: ACM, 2010, 328-332. <https://doi.org/10.1145/1755688.1755732>
  56. Patil, M. D., Sane, S. S. Effective Classification after Dimension Reduction: A Comparative Study. International Journal of Scientific and Research Publications, 2014, 4(7), 1-4.
  57. Peng, H., Gates, C., Sarma, B., Li, N., Qi, Y., Potharaju, R., Nita-Rotaru, C., Molloy, I. Using Probabilistic Generative Models for Ranking Risks of Android Apps. In Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS'12), Raleigh, North Carolina, USA, 2012, 241-252. <https://doi.org/10.1145/2382196.2382224>
  58. Platt, J. C. Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. Advances in Kernel Methods, 1998. <https://doi.org/10.1.1.43.4376>
  59. Popper, B. Google Announces over 2 Billion Monthly Active Devices on Android, 2017. Retrieved January 1, 2019, from <https://www.theverge.com/2017/5/17/15654454/android-reaches-2-billion-monthly-active-users>
  60. Portokalidis, G., Homburg, P., Anagnostakis, K., Bos, H. Paranoid Android: Versatile Protection For Smartphones. In Annual Computer Security Applications Conference (ACSAC), Austin, Texas, USA, 2010, 347-356. <https://doi.org/10.1145/1920261.1920313>
  61. pxb1988/dex2jar: Tools to Work with Android .dex and java .class Files, 2018. Retrieved January 1, 2019, from <https://github.com/pxb1988/dex2jar>
  62. Rastogi, V., Chen, Y., Enck, W. AppsPlayground: Automatic Security Analysis of Smartphone Applications. In CODASPY '13 Proceedings of the third ACM conference on Data and Application Security and Privacy, San Antonio, Texas, USA, 209-220. <https://doi.org/10.1145/2435349.2435379>
  63. Requesting Permissions at Run Time, 2018. Retrieved April 1, 2018, from <https://developer.android.com/training/permissions/requesting.html>



64. Roy, K., Kar, S., Das, R. N. Understanding the Basics of QSAR for Applications in Pharmaceutical Sciences and Risk Assessment. Elsevier Science, 2015. <https://doi.org/10.1016/C2014-0-00286-9>
65. Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Bringas, P. G. On the Automatic Categorisation of Android Applications. In 2012 IEEE Consumer Communications and Networking Conference (CCNC 2012), Las Vegas, NV, USA, 2012, 149-153. <https://doi.org/10.1109/CCNC.2012.6181075>
66. Sayfullina, L., Eirola, E., Komashinsky, D., Palumbo, P., Miche, Y., Lendasse, A., Karhunen, J. Efficient Detection of Zero-Day Android Malware Using Normalized Bernoulli Naive Bayes. In Proceedings of the 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom 2015), Helsinki, Finland, 2015, 198-205. <https://doi.org/10.1109/Trustcom.2015.375>
67. Security Center - Overview, 2018. Retrieved April 1, 2018, from <https://www.android.com/security-center/>
68. Security Tips, 2018. Retrieved April 29, 2018, from <https://developer.android.com/training/articles/security-tips#RequestingPermissions>
69. Shabtai, A., Tenenboim-Chekina, L., Mimran, D., Rokach, L., Shapira, B., Elovici, Y. Mobile Malware Detection Through Analysis of Deviations in Application Network Behavior. *Computers Security*, 2014, 43, 1-18. <https://doi.org/10.1016/j.cose.2014.02.009>
70. Shaun Aimoto, 2016. Five Ways Android Malware is Becoming More Resilient. Retrieved May 6, 2018, from <https://www.symantec.com/connect/blogs/five-ways-android-malware-becoming-more-resilient>
71. Shevade, S. K., Keerthi, S. S., Bhattacharyya, C., Murthy, K. R. K. Improvements to the SMO Algorithm for SVM Regression. *IEEE Transactions on Neural Networks*, 2000, 11(5), 1188-1193. <https://doi.org/10.1109/72.870050>
72. Singh, P., Tiwari, P., Singh, S. Analysis of Malicious Behavior of Android Apps. *Procedia Computer Science*, 2016, 79, 215-220. <https://doi.org/10.1016/j.procs.2016.03.028>
73. Smartphone OS Global Market Share 2009-2017, 2018. Retrieved July 7, 2018, from <https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/>
74. Suarez-Tangil, G., Dash, S. K., Holloway, R., Ahmadi, M., Giacinto, G., Kinder, J., Cavallaro, L. DroidSieve: Fast and Accurate Classification of Obfuscated Android Malware. Proceedings of the 7th ACM Conference on Data and Application Security and Privacy (CODASPY 2017), Scottsdale, Arizona, USA, 2017, 309-320. <https://doi.org/10.1145/3029806.3029825>
75. Suarez-Tangil, G., Tapiador, J. E., Peris-Lopez, P., Blasco, J. Dendroid: A Text Mining Approach to Analyzing and Classifying Code Structures in Android Malware Families. *Expert Systems with Applications*, 2014, 41(4 PART 1), 1104-1117. <https://doi.org/10.1016/j.eswa.2013.07.106>
76. Suarez-Tangil, G., Tapiador, J. E., Peris-Lopez, P., Ribagorda, A. Evolution, Detection and Analysis of Malware for Smart Devices. *IEEE Communications Surveys and Tutorials*, 2014, 16(2), 961-987. <https://doi.org/10.1109/SURV.2013.101613.00077>
77. Tam, K., Feizollah, A., Anuar, N. B., Salleh, R., Cavallaro, L. The Evolution of Android Malware and Android Analysis Techniques. *ACM Computing Surveys*, 2017, 49(4), 1-41. <https://doi.org/10.1145/3017427>
78. The Judy Malware: Possibly the Largest Malware Campaign Found on Google Play, 2017. Retrieved December 26, 2018, from <https://blog.checkpoint.com/2017/05/25/judy-malware-possibly-largest-malware-campaign-found-google-play/>
79. Wain, K., Au, Y., Zhou, Y. F., Huang, Z., Lie, D. PScout: Analyzing the Android Permission Specification. In CCS '12 Proceedings of the 2012 ACM Conference on Computer and Communications Security, Raleigh, North Carolina, USA: ACM, 2012, 217-228. <https://doi.org/10.1145/2382196.2382222>
80. Wang, C., Li, Z., Mo, X., Yang, H., Zhao, Y. An Android Malware Dynamic Detection Method Based on Service Call Co-occurrence Matrices. *Annals of Telecommunications*, 2017, 72(9-10), 1-9. <https://doi.org/10.1007/s12243-017-0580-9>
81. Wang, Y., Zheng, J., Sun, C., Mukkamala, S. Quantitative Security Risk Assessment of Android Permissions and Applications. In Wang, L., Shafiq, B. (Eds.), 27th Data and Applications Security and Privacy (DBSec), Newark, NJ, USA: Springer, 2013, 226-241. [https://doi.org/10.1007/978-3-642-39256-6\\_15](https://doi.org/10.1007/978-3-642-39256-6_15)
82. Wei, X., Gomez, L., Neamtiu, I., Faloutsos, M. Malicious Android Applications in the Enterprise: What Do They Do and How Do We Fix It? In ICDEW '12 Proceedings of the 2012 IEEE 28th International Conference on Data Engineering Workshops, Arlington, Virginia, USA: IEEE, 2012, 251-254. <https://doi.org/10.1109/ICDEW.2012.81>
83. Wu, D.-J., Mao, C.-H., Wei, T.-E., Lee, H.-M., Wu, K.-P. DroidMat: Android Malware Detection through Mani-

- fest and API Calls Tracing. In 2012 Seventh Asia Joint Conference on Information Security, Minato, Tokyo, Japan, 2012, 62-69. <https://doi.org/10.1109/AsiaJ-CIS.2012.18>
84. Xue, Y., Meng, G., Liu, Y., Tan, T. H., Chen, H., Sun, J., Zhang, J. Auditing Anti-Malware Tools by Evolving Android Malware and Dynamic Loading Technique. *IEEE Transactions on Information Forensics and Security*, 2017, 12(7), 1529-1544. <https://doi.org/10.1109/TIFS.2017.2661723>
85. Yang, M., Wang, S., Ling, Z., Liu, Y., Ni, Z. Detection of Malicious Behavior in Android Apps Through API Calls and Permission Uses Analysis. *Concurrency and Computation: Practice and Experience*, 2017, 29(19), 1-13. <https://doi.org/10.1002/cpe.4172>
86. Yerima, S. Y., Sezer, S., McWilliams, G., Muttik, I. A New Android Malware Detection Approach Using Bayesian Classification. In 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA), Barcelona, Spain: IEEE, 2013, 121-128. <https://doi.org/10.1109/AINA.2013.88>
87. Yu, J., Huang, Q., Yian, C. H. DroidScreening: A Practical Framework for Real-World Android Malware Analysis. *Security and Communication Networks*, 2016, 9(11), 1435-1449. <https://doi.org/10.1002/sec.1430>
88. Yuan, Z., Lu, Y., Xue, Y. DroidDetector: Android Malware Characterization and Detection Using Deep Learning. *Tsinghua Science and Technology*, 2016, 21(1), 114-123. <https://doi.org/10.1109/TST.2016.7399288>
89. Zheng, M., Sun, M., Lui, J. C. S. DroidTrace: A Ptrace Based Android Dynamic Analysis System with Forward Execution Capability. In *IWCMC 2014 - 10th International Wireless Communications and Mobile Computing Conference*, Nicosia, Cyprus, 2014, 128-133. <https://doi.org/10.1109/IWCMC.2014.6906344>
90. Zhou, Y., Jiang, X. Dissecting Android Malware: Characterization and Evolution. In *Proceedings of the 33rd IEEE Symposium on Security and Privacy*, San Francisco, CA, USA: IEEE, 2012, 95-109. <https://doi.org/10.1109/SP.2012.16>
91. Zhu, H. J., You, Z. H., Zhu, Z. X., Shi, W. L., Chen, X., Cheng, L. DroidDet: Effective and Robust Detection of Android Malware Using Static Analysis Along with Rotation Forest Model. *Neurocomputing*, 2017, 272, 638-646. <https://doi.org/10.1016/j.neucom.2017.07.030>