SAFEDroid: Using Structural Features for Detecting Android Malwares

Sevil Sen[1], Ahmet I. Aysan[1], and John A. Clark[2]

[1] Hacettepe University
ssen@cs.hacettepe.edu.tr, aysan@hacettepe.edu.tr,
[2] University of Sheffield,
john.clark@sheffield.ac.uk

Abstract. Mobile devices have become a popular target for attackers, whose aims are to harm the devices, illegally obtain personal information and ultimately to reap financial benefit. In order to detect such malicious attempts, security solutions based on static analysis are mainly preferred due to resource-constraints of these devices. However, in general, static analysis-based solutions are not very effective against new mobile malwares and new variants of existing mobile malwares appear on a daily basis. In this study, new features for static analysis are investigated in order to detect mobile malwares. While studies found in the literature mostly employ API calls and permissions, this current study explores some novel structural features. Results show the relative effectiveness of these features on malware detection. Furthermore, it is shown that these features detect new malwares better than solely applying API-based features.

Keywords: Android security, malware detection, static analysis, structural features, machine learning

1 INTRODUCTION

Android malware is one of the biggest threats today. With mobile devices having become an integral part of modern lifes, attackers focus more and more on harming mobile devices and stealing private information from them. New mobile malwares are emerging every day. Kaspersky [1] reported that 884,774 new malicious applications appeared in 2015 alone, three times more than the number of new mobile malwares seen in 2014. According to the recent McAfee Labs Threats Report [2], the number of mobile malwares is still on the rise.

There are two main approaches to the detection of mobile malwares : static analysis and dynamic analysis. Static analysis is generally preferred over dynamic analysis due to its lower overhead on mobile devices. Therefore, a significant amount of work has been proposed on static-based malware detection for Android devices over the last five years, and different features have been investigated in different approaches.

In the literature, permissions and API calls are among the most used features for static analysis [3]. However, the structural features of an application have not been explored as part of the proposed approaches, hence it forms the main aim of this current study. It can be seen from the literature that the structural information of portable executables has a positive impact on malware detection [4]. Therefore, this current study investigates the use of structural features of applications on mobile malware detection. To the best of our knowledge, some of these proposed features are novel and have not been the subject of previous studies. The proposed system is also applied to new malwares in order to see its capability of detecting new malwares and new variants of malwares. In recent years, there has been a significant increase in Android malware variants, much more than new malware families [5]. Therefore, detecting variants of existing malwares is an important characteristic of an anti-malware system.

To summarize, the contributions of this paper are as follows :

- SAFEDroid, an Android malware detection system based on static analysis, is introduced.
- Rigorous analysis of the original features of SAFEDroid is conducted in order to increase the accuracy of the system.
- SAFEDroid is evaluated on new variants of existing malwares as well as new malwares in order to demonstrate the impact of the original features employed by the proposed system.

The remainder of this paper is organized as follows. Section 2 discusses the related works in the literature, and outlines existing malware detection systems based on static analysis together with the features they employ. Section 3 introduces the features and classification algorithms employed in this study. Section 4 evaluates the results and also discusses the limitations of the proposed approach. Finally, the study is concluded in Section 5.

2 RELATED WORK

Static analysis is the most employed technique on mobile devices due to its efficiency. One of the early studies relying on static analysis is by Kirin [6] which detects certain types of malware by evaluating the configuration of an application against a collection of security rules. These rules are defined by using permissions and action strings in Intent filters extracted from the manifest file. Even though it is not a complete solution for the problem, it was shown that it could mitigate certain types of malware. Stowaway [7] detects overprivileged apps by analyzing API Calls. ComDroid [8] detects application' vulnerabilities by analyzing inter-application communications; with their analysis showing that Android applications are vulnerable to attack by other applications.

RiskRanker [9] proposed a two-level analysis. High-risk and medium-risk applications are determined in the first-order analysis, and applications employing obfuscating, encryption or dynamic class loading techniques are extracted among

these risky applications in the second-order analysis. Only static analysis is employed to identify applications employing suspicious behaviors. DroidAnalyzer [10] employs static analysis in order to detect the presence of root exploits, and identifies the potential risks of related apps. Droid Analytics [11] generate signatures at the app/class/method level, hence it can detect 0-day repackaged malwares.

In another approach for malware detection, Drebin [12] uses hardware and app components, permissions, API calls and network address as features in order to apply machine learning techniques. Drebin not only classifies the applications, but also produces explanations for the detection model. Similarly, DroidAPIMiner [13] achieves a high detection rate and low false positive rate by using kNN classifier on API calls. They firstly remove API calls invoked by third-party packages, and then consider only top APIs with the highest difference between malware and benign apps in training. Therefore, it is not susceptible to evasive attacks that add more benign APIs to the code in order to evade detection. SafeDroid [14] is also based on the top distinguished API calls extracted from dex files. Although our study has used the same acronym as SafeDroid [14], it is an independent work which evaluates not only the effect of API calls but also the effect of structural features on malware detection. This replication occurred purely coincidentally without realization on the part of the authors. Another ML-based approach firstly groups applications with similar functionalities in clusters, then employs kNN classifier in order to detect mobile malwares [15]. The performance of the approach is shown to be better than AndroGuard [16]; however, it does not perform well on the detecting of updated attacks such as BaseBridge and DroidKungFu.

The usage of ML-based techniques for mobile malware detection is increasing. One of the recent studies employs Bayesian classifiers to distinguish malicious from benign applications [17]. They extract features based on API calls, system and Android commands, permissions in the manifest file, and other information such as the usage of encryption, and the presence of a second .apk file. Mutual Information calculation is employed to rank features, and as a result the top 25 features are outlined and used in training. They also show that as the training sample set increases, the performance of the classifier on the same feature set also increases. A recent study employed different machine learning techniques, including deep learning to the problem [18]. Another recent work evaluates ML-based mobile malware detectors [19].

One of the studies in the literature shows similarity to this current study, employing data mining algorithms on static features obtained from apk, dex, and XML files of an application [20]. However, due to the unavailability in 2010 of malware datasets, the proposed model was not applied for malware classification. Its evaluation is carried out to differentiate between game and tool applications. Moreover, only one feature from [20] is used common to this current study.

To summarize, the studies found in the literature mainly use permissions and API calls for static analysis-based malware detection. Application metadata has also been proposed as complementary to static and dynamic analysis [25].

Static Tools	Purpose	Features
Kirin [6]	mitigating certain types of malwares	permissions action strings
Stowaway [7]	determining overprivilege	API calls action strings
ComDroid [8]	detecting apps communication vulnerabilities	permissions intents components
RiskRanker [9]	risk analysis for detecting 0-day malwares	data flow analysis control flow analysis suspicious activities
DroidMat [15]	detecting malwares	permissions intents components API calls
Droid Analytics [11]	generating signatures detecting 0-day repackaged malwares	API calls
DroidAPIMiner [13]	detecting malwares	API calls
Bayesian [17]	detecting malwares	permissions API calls commands
Drebin [12]	detecting malwares producing explanations for the model	permissions intents components API calls network addresses
DroidAnalyzer [10]	detecting the presence of root exploits	API calls keywords
Anastasia [18]	detecting malwares	permissions intents API calls system commands malicious activities
ML-based detectors [19]	detecting malwares evaluating ml-based detectors	basic blocks from CFG
Resource Usage-based [21]	detecting malwares	resource usage
Metadata-based [22]	detecting malwares	permissions API calls metadata
ADRoit [23]	detecting malwares	permissions metadata
AndroDialysis [24]	detecting malwares	permissions intents

Table 1. Existing Android Static Analysis Tools

Recently, studies that use metadata for malware detection [23][22] have been introduced. A brief survey in [3] provides further information on the features used for mobile malware detection. The static analysis tools employed in the literature are outlined and sorted according to their publication date in Table 1.

3 METHODOLOGY

In this study, a static analysis tool is developed that is based on API calls and the structural features of applications. Figure 1 illustrates a simplified system architecture of SAFEDroid.

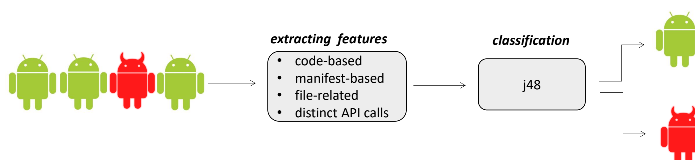

Fig. 1. Simplified schema of the proposed approach

First of all, a dataset consisting of malicious and benign malwares is constructed. Then, a rigorous study is conducted on the dataset in order to select the right features to increase the detection module's accuracy. The choice of which characteristics of an application can be used for machine learning is very important, and must contain sufficient information to allow the fundamentals to be developed. However, irrelevant or an excessive number of features could degrade the performance of the learning algorithms. Based on feature analysis, the feature subset, which is believed to discriminate malicious applications from goodwares, is selected and extracted. Machine learning techniques are applied for the classification of applications by using this feature subset. Finally, the model produced is evaluated by using a new dataset obtained from Drebin [12]. The results support the researchers' hypotheses that the structural features of an application are indicative of malicious behavior, and could detect new malware variants. Each step of the proposed approach and the experimental results are presented in the subsequent sections.

3.1 Feature Selection

In the literature, API calls and permissions of applications are features that are extensively employed for static analysis in mobile malware detection [3]. It is shown that the systems relying on API calls-based features achieve better

detection performance than systems using a permission-based feature set [13]. Moreover, the usage of API and permission-based features together does not engender improvements over the usage of only API-based features [26]. Therefore, this study has not included permission features; the top distinguished API calls have been added to the features used. In addition to API calls, the proposed system in this study also extracts some structural features of an application based on analysis of malwares and goodwares. The features employed in this study are classified into three groups : code-based, manifest-based, and file-related features. All of these features are summarized in Table 2.

Code-Based Features The features in this group are extracted from the application code. The following three features in this group have previously been used for the detection of malwares in other studies : DexClassLoader, Crypto API usage and, API calls. As far as the researchers of this study are aware, the other features are being employed for the first time in this study.

- **DexClassLoader** Android gives developers an opportunity to load and use classes in runtime. These files can be dynamically loaded from a remote server or from any path of the device. Surprisingly, according to analysis, the percentage usage of dynamic code loading in the malware dataset is much less than its usage in the benign dataset. It was noted that almost half of the benign applications use dynamic code loading. This result is also consistent with the analysis of one million apps submitted to Andrubis [27]. There is a substantial increase in the number of applications using the updating techniques, especially dynamic class loading [28]. It could be concluded that dynamic code loading alone may no longer be an indicator for malicious behavior due to its rising popularity among goodwares as well.
- **Crypto API** Attackers mainly use cryptography in order to evade static analysis. On the other hand, its usage statically among goodwares has also increased over the last few years [27]. Analysis conducted in this study also supports the increasing usage of crypto API in benign applications.
- **Goto** The *goto* statement could be inserted into code in order to evade signature-based detection tools. By using goto statements, attackers can change the order of the code, but preserve the code execution sequence at runtime.
- **Annotation** Annotations are types of metadata that can be added to the code, but have no effect on the runtime. Our analysis shows that benign applications tend to use many more annotations than malicious applications.
- **Methods** This feature represents the number of methods in an application package. Benign applications appear to use many more methods than malwares.
- **Classes** This feature represents the number of classes in an application package. The number of classes seems to be much higher in benign applications than malwares.
- **Used Permissions** The application authors generally request more permissions than they use. Since used permissions give more information about an

application, the number of used permissions are also taken into account in this study. PScout's API-permission list [29] are used in order to extract the real permission list of applications. Recently, a few studies in the literature analyze and employ the combination of used permissions for malware detection [30][31][32]. However, to the best of our knowledge, the number of used permissions employed in this current study is a first.
- **Used Dangerous Permissions** This feature represents the used dangerous permissions by an application and to the best of knowledge of the researchers of this study it is also a first to be employed.
- **API Call** The top distinguished API calls, whose usage in the malware dataset is higher than the benign dataset, as in DroidAPIMiner [13], are extracted.

Manifest-Based Features Every Android application must have a manifest file (*AndroidManifest.xml*) that contains essential information about the application such as information about the package and application components such as activities, services, broadcast receivers, and content providers. In this study, only the number of permissions listed in the file, and the size of the manifest file is used.

- **Permissions** This feature represents the number of permissions requested in the manifest file. According to analysis in this study, benign applications use 8 permissions on average, whereas malwares use 14 permissions. Attackers generally use more permissions in order to obtain the control of the device. They can also obtain permissions in advance for use in the dynamically loaded code at runtime [33]. Permissons are the most used features in static analysis [3]. While studies mainly extract the permissions listed in the manifest file [18][20], to the best of knowledge of the researchers the number of permissions is not explicitly used as a feature. Only in [34] the count of xml elements in the manifest file besides permissions is also considered.
- **Dangerous Permissions** This feature represents the number of dangerous permissions requested in the manifest file. It is shown that malwares generally request more dangerous permissions than benign apps [35].
- **Lines** Another important feature about the manifest file is the size of the file which is evaluated by the number of lines here. The researchers observed that benign applications mostly generate well-written manifest files with more information. Research experimentation for this study revealed that the average line count of the manifest files of goodwares is 81, whereas it is 49 for malwares.

File-Related Features Android application is similar to *jar* file and it contains class files and hierarchical directories. Besides, resource files used by applications are stored in the *resource* directory. These features might have a positive effect in order to classify samples. To the best of our knowledge, the file-related features have not been included in previous studies.

Category	Features	Descriptions	Previous Works
Code-Based	DexClassLoader	DexClassLoader API Usage	Yes
	Crypto API	cryptographic API usage	Yes
	Goto	number of goto statements	No
	Annotation	number of annotations	No
	Methods	number of methods	No
	Classes	number of classes	No
	Used Permissions	number of used permissions	No
	Used Dangerous Permissions	number of used dangerous permissions	No
	API Calls	API calls usage	Yes
Manifest-Based	Permissions	number of requested permissions	Yes [34]
	Dangerous Permissions	number of requested dangerous permissions	No
	Lines	number of lines of the manifest file	No
File-Related	File Size	size of the application (bytes)	Yes [20]
	Files	number of files	No
	Directories	number of directories	No
	Resource Files	number of files stored in resource directory	No

Table 2. Category of features

- **File Size** This feature represents the file size of an application measured in bytes. The researchers found that benign applications generally have larger file sizes than malicious applications. In the literature, this feature has been used to differentiate game applications from tools and, has been suggested for use in malware detection [36].
- **Files** This feature represents the number of files in an application package. Benign applications appear to have many more files than malwares.
- **Directories** This feature represents the number of directories in an application package. It is observed that benign applications have many more directories in their packages.

– **Resource Files** This feature indicates the number of resource files used by an application. The resources in Android could be used for various reasons such as language support and, providing images for UI. Applications in the benign dataset are observed to use more resources than those in the malware dataset.

When the correlation between the features and the class labels (i.e. benign or malicious) is analyzed, all non-zeros values are observed. The smallest absolute correlation value (0.14) belongs to *Lines* feature. However when principle component analysis (PCA) is carried out, it is seen that even this feature has considerable weight for the eigen vector corresponding to the 5th highest eigen value. When the correlation among the features are analyzed, it is observed that some features are highly correlated as expected, such as *Methods* and *Classes*. However in general, the cross-correlation matrix shows that the feature set exhibits low inter-dependency, even with feature couples having zero correlation, such as *DexClassLoader* and *Dangerous Permissions*.

3.2 Classification

All applications are firstly disassembled into .smali files using Apktool [37] and then features are extracted. In this stage well-known machine learning algorithms are applied in order to classify applications as either benign or malicious based on these features. A decision tree algorithm is used that is named J48 [38], which is an implementation of the C4.5 algorithm. Weka tool [39] is used in the application of the C4.5 algorithm for this study. Other well-known machine learning algorithms such as kNN and SVM are also evaluated in this study. However since J48 produces the highest level of accuracy, it was elected to be employed across all other relevant experiments of this study. J48 is one of the top ten data mining algorithms [40], which also allows us for interpretation of the tree in order to see the features that are best separating malicious and benign applications.

In training, MalGenome dataset is used for representing malicious applications. This dataset contains 1260 malwares from 49 malware families. A benign dataset consisting of 1260 applications downloaded from Google Play was also created. A tool based on Android Market API [41] was utilized in obtaining the benign samples from Google Play. Particular attention was paid to select applications downloaded more than 5 million times to ensure that they were not malicious. Furthermore, they were checked with VirusTotal [42] in order to ensure these samples are not malware. Applications are only included in the dataset if they are not detected by any of the antivirus solutions. Hence the benign dataset is reduced to 978 applications. However the benign dataset is aimed to be extended in the future. In order to see whether or not a generated model could detect new attacks, an evaluation of the models using the Drebin dataset [12] was also conducted. This public dataset contains 5,560 applications from 179 different malware families. Hence, it introduces new malware families that do not exist in MalGenome; however it also contains some samples

from MalGenome too. In the experiments conducted for this study, the malware samples also common to MalGenome were extracted to specifically evaluate the detection performance of the generated models on unknown malwares.

4 EVALUATION

In order to evaluate the performance of each model, the following metrics were employed: true positive ratio; false positive ratio; and accuracy. True positive ratio shows the ratio of correctly classified malicious applications (TP: true positives) to all malicious applications in the dataset. False positive ratio represents the misclassified applications as malicious (FP: false positives) to all benign applications in the dataset. TN represents true negatives, FN represents false negatives in the equations below.

$$True\ Positive\ Ratio = \frac{(TP)}{(TP + FN)} \tag{1}$$

$$False\ Positive\ Ratio = \frac{(FP)}{(TN + FP)} \tag{2}$$

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{3}$$

Firstly the classifiers were trained by using only API calls which are shown to be very effective against detecting mobile malwares [13]. The effect of the number of API calls can be seen in Figure 2. The result seen is consistent with DroidAPIMiner's outcome in that 169 API calls produce the highest accuracy. Therefore, 169 API calls were employed for the remainder of the experiments. The top 20 API calls having the highest difference between malware and be-

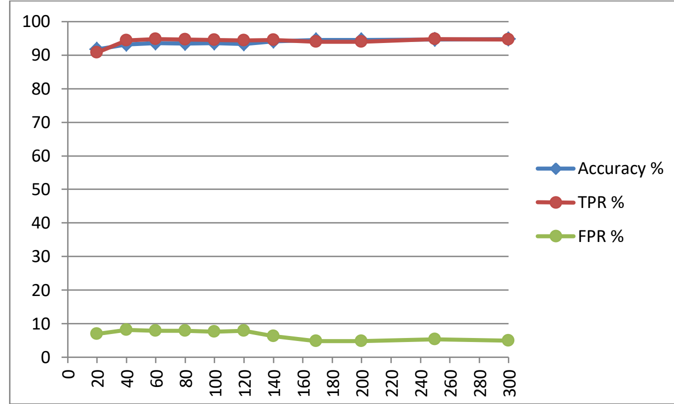

Fig. 2. Effect of the number of API calls

nign datasets are also given in Figure 3. The top 20 APIs shows similarity with DroidAPIMiner's results [13], for which analysis was performed on a larger dataset.

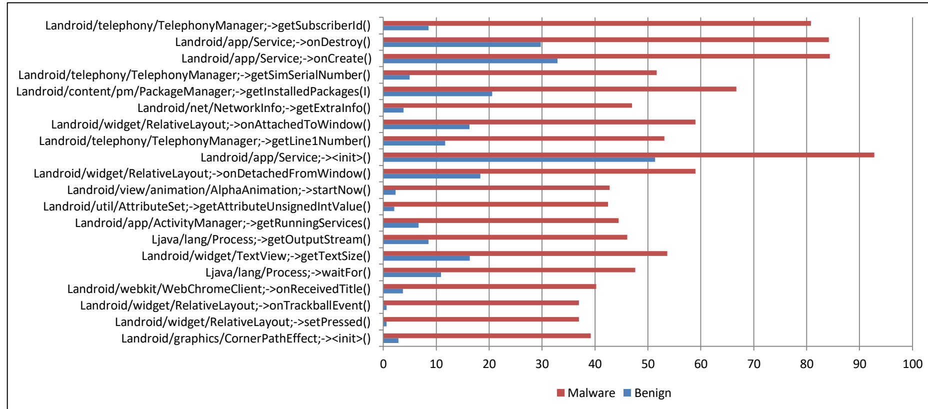

Fig. 3. The top 20 distinguishable API calls

Classifiers were also trained using different combinations of features and the results compared. F1 represents the code-based features except for API calls, F2 represents manifest-based features, F3 is used for file-related features, and F4 indicates the most distinguishable API calls. Based on the outcome of DroidAPIMiner [13], 169 API calls were employed. Two training schemes are employed : cross-validation and 66% percentage split.

Figure 4 shows the percentage of correctly classified applications. As can be seen, manifest-based and file-related features do not perform as well as code-based features in distinguishing malicious from benign applications. The combination of code-based features and API calls produces the lowest error rate. The performance of this combination is quite close to the combination of all features. Hence, both combinations are evaluated on the new malwares. The detection and false positive ratios of both combinations are exhibited in Table 3.

Feature Sets	**DR**	**FPR**
F14: Code-based and API calls	98.4%	1.8%
FAll: All	98.3%	2.0%

Table 3. Performance of classifiers

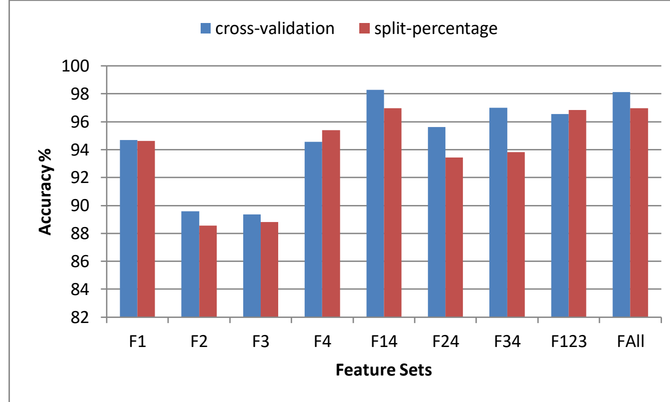

Fig. 4. Percentage of correctly classified instances

Finally, in order to see the performance of this approach against new variants of existing malwares and new malwares, evaluation was also conducted against the Drebin dataset [12], which is a larger dataset than MalGenome [43]. It should be noted that, all applications that exist in MalGenome were removed from the test dataset before the evaluation took place. The results are presented in Table 4. In particular, the malware families considered in training are shown. The results show that the proposed approach effectively detects new variants of existing malwares. While API-based approach could not detect some families at all, the new features introduced here make them distinguishable from benign applications. On the other hand, the proposed approach is ineffective against Plankton family, which is a type of update attack. This family could not be detected effectively by applying API-based features as well. Other than Plankton family, we could say that the proposed approach considerably increases the accuracy against new variants of existing malwares and new malware families.

4.1 Limitations

An attacker who knows that the device is protected with SAFEDroid could change its code while preserving its malicious behavior in order to evade the system. For example, it could increase the size of the manifest file and the file size in order to look less suspicious. Actually, it is the common vulnerability for any detection system based on static analysis. Attackers always employ evasive techniques to be successfully installed and run on the device. It is the nature of an arms race mechanism between virus and antivirus systems. It should however be emphasized that the attacker has to change critical features employed in this study in order to evade SAFEDroid, which could cause both a decrease in the impact and an increase in the cost of the attack. For example, an attacker could change the number of permissions requested in the manifest file, and seem non-overprivileged, but then the attacker might not be able to run dynamically loaded

Family	Family Size	API	Code-based and API	All
Adrd	24	91.67%	91.67%	91.67%
BaseBridge	22	63.64%	63.64%	72.73%
Bgserv	1	0.00%	0.00%	100.00%
DroidDream	34	79.41%	94.12%	94.12%
DroidKungFu	193	90.76%	%94.81	%95.34
FakePlayer	11	0.00%	90.91%	90.91%
Geinimi	25	80.00%	91.30%	91.30%
GGTrack	2	0.00 %	100.00%	100.00%
GinMaster	338	91.41%	96.15%	96.15%
GPSpy	2	0.00%	100.00%	100.00%
JiFake	28	0.00%	89.29%	89.29%
KMin	96	100.00%	100.00%	100.00%
Nickspy	9	77.78%	88.89%	88.89%
Plankton	614	29.48%	2.12%	14.50%
Spitmo	10	100.00%	100.00%	100.00%
TapSnake	2	50.00%	100.00%	100.00%
Yzhc	15	100.00%	100.00%	100.00%
Zitmo	13	69.23%	76.92%	76.92%
Zsone	2	50%	100%	100%
DREBIN	4432	**42.95%**	**70.21%**	**61.86%**

Table 4. Detection ratio on new malwares

code at runtime. An attacker could also uses less *goto* statements by not using evasion techniques such as code reordering, but then might need to use other evasion techniques in order to hide its malicious code. Furthermore, an attacker could not evade API-based features by increasing the size of benign features, since API calls considered here are the most distinguishable API calls from benign applications in malware applications [13]. To summarize, an attacker, who pursues to achieve his goals no matter what happens, could avoid detection by SAFEDroid by changing the features, but that would come with a cost for the attacker.

5 CONCLUSIONS

There have been many studies proposed for malware detection on mobile devices. Many of these approaches are based on static analysis due to resource constraints of these devices. Those approaches mainly employe API calls and permissions as features. In this current study, structural features such as the number of methods/classes, the size of the application, and the number of *goto* statements are explored for the purpose of malware detection. In particular, three groups of features are analyzed : code-based, manifest-based, and file-related. The analysis shows that code-based features together with API calls achieve a high detection rate with a low false positive ratio. Furthermore, these novel features are shown

to be effective against new malwares and new variants of malwares. The results obtained were an improvement on solely applying API-based features.

6 ACKNOWLEDGEMENTS

This study is supported by the Scientific and Technological Research Council of Turkey (TUBITAK-115E150). We would like to thank TUBITAK for its support.

References

1. Kaspersky Lab. The volume of new mobile malware tripled in 2015, March 2016. http://www.kaspersky.com/about/news/virus/2016/The_Volume_of_New\newline_Mobile_Malware_Tripled_in_2015.
2. McAfee Lab. Mcafee lab threats report, June 2016. https://www.mcafee.com/us/resources/reports/rp-quarterly-threats-jun-2017.pdfhttps://www.mcafee.com/us/resources/reports/rp-quarterly-threats-jun-2017.pdf.
3. Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, and Ainuddin Wahid Abdul Wahab. A review on feature selection in mobile malware detection. *Digital Investigation*, 13:22–37, 2015.
4. M Zubair Shafiq, S Momina Tabish, Fauzan Mirza, and Muddassar Farooq. Pe-miner: Mining structural information to detect malicious executables in realtime. In *International Workshop on Recent Advances in Intrusion Detection*, pages 121–141. Springer, 2009.
5. Symantec. Internet security threat report, April 2016. https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf.
6. William Enck, Machigar Ongtang, and Patrick McDaniel. On lightweight mobile phone application certification. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 235–245. ACM, 2009.
7. Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 627–638. ACM, 2011.
8. Erika Chin, Adrienne Porter Felt, Kate Greenwood, and David Wagner. Analyzing inter-application communication in android. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 239–252. ACM, 2011.
9. Michael Grace, Yajin Zhou, Qiang Zhang, Shihong Zou, and Xuxian Jiang. Riskranker: scalable and accurate zero-day android malware detection. In *Proc. of the 10th international conference on Mobile systems, applications, and services*, pages 281–294. ACM, 2012.
10. Seung-Hyun Seo, Aditi Gupta, Asmaa Mohamed Sallam, Elisa Bertino, and Kang-bin Yim. Detecting mobile malware threats to homeland security through static analysis. *Journal of Network and Computer Applications*, 38:43–53, 2014.
11. Min Zheng, Mingshen Sun, and John CS Lui. Droid analytics: a signature based analytic system to collect, extract, analyze and associate android malware. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on*, pages 163–171. IEEE, 2013.

12. Daniel Arp, Michael Spreitzenbarth, Malte Hübner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS)*, 2014.
13. Yousra Aafer, Wenliang Du, and Heng Yin. Droidapiminer: Mining api-level features for robust malware detection in android. In *Security and Privacy in Communication Networks*, pages 86–103. Springer, 2013.
14. Rohit Goyal, Angelo Spognardi, Nicola Dragoni, and Marios Argyriou. Safedroid: A distributed malware detection service for android. In *Service-Oriented Computing and Applications (SOCA), 2016 IEEE 9th International Conference on*, pages 59–66. IEEE, 2016.
15. Dong-Jie Wu, Ching-Hao Mao, Te-En Wei, Hahn-Ming Lee, and Kuo-Ping Wu. Droidmat: Android malware detection through manifest and api calls tracing. In *Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on*, pages 62–69. IEEE, 2012.
16. Androguard, March 2017. `https://github.com/androguard/androguard/`.
17. Suleiman Y Yerima, Sakir Sezer, Gavin McWilliams, and Igor Muttik. A new android malware detection approach using bayesian classification. In *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*, pages 121–128. IEEE, 2013.
18. Hossein Fereidooni, Mauro Conti, Danfeng Yao, and Alessandro Sperduti. Anastasia: Android malware detection using static analysis of applications. In *New Technologies, Mobility and Security (NTMS), 2016 8th IFIP International Conference on*, pages 1–5. IEEE, 2016.
19. Kevin Allix, Tegawendé F Bissyandé, Quentin Jérome, Jacques Klein, Yves Le Traon, et al. Empirical assessment of machine learning-based malware detectors for android. *Empirical Software Engineering*, 21(1):183–211, 2016.
20. Asaf Shabtai, Yuval Fledel, and Yuval Elovici. Automated Static Code Analysis for Classifying Android Applications Using Machine Learning. *2010 International Conference on Computational Intelligence and Security*, pages 329–333, December 2010.
21. Gerardo Canfora, Eric Medvet, Francesco Mercaldo, and Corrado Aaron Visaggio. Acquiring and analyzing app metrics for effective mobile malware detection. In *Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics*, pages 50–57. ACM, 2016.
22. Tao Ban, Takeshi Takahashi, Shanqing Guo, Daisuke Inoue, and Koji Nakao. Integration of multi-modal features for android malware detection using linear svm. In *Information Security (AsiaJCIS), 2016 11th Asia Joint Conference on*, pages 141–146. IEEE, 2016.
23. Alejandro Martín, Alejandro Calleja, Héctor D Menéndez, Juan Tapiador, and David Camacho. Adroit: Android malware detection using meta-information. In *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*, pages 1–8. IEEE, 2016.
24. Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, Guillermo Suarez-Tangil, and Steven Furnell. Androdialysis: Analysis of android intent effectiveness in malware detection. *Computers & Security*, 65:121–134, 2017.
25. Peter Teufl, Michaela Ferk, Andreas Fitzek, Daniel Hein, Stefan Kraxberger, and Clemens Orthacker. Malware detection by applying knowledge discovery processes to application metadata on the android market (google play). *Security and Communication Networks*, 2013.

26. Ahmet Ilhan Aysan and Sevil Sen. Api call and permission based mobile malware detection. In *Signal Processing and Communications Applications Conference (SIU), 2015 23th*, pages 2400–2403. IEEE, 2015.
27. Martina Lindorfer, Matthias Neugschwandtner, Lukas Weichselbaum, Yanick Fratantonio, Victor Van Der Veen, and Christian Platzer. Andrubis-1,000,000 apps later: A view on current android malware behaviors. In *Proceedings of the the 3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, 2014.
28. Ahmet Ilhan Aysan and Sevil Sen. "Do you want to install an update of this application?" A rigorous analysis of updated android applications. In *Cyber Security and Cloud Computing (CSCloud), 2015 IEEE 2nd International Conference on*, pages 181–186. IEEE, 2015.
29. Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. Pscout: analyzing the android permission specification. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 217–228. ACM, 2012.
30. Veelasha Moonsamy, Jia Rong, and Shaowu Liu. Mining permission patterns for contrasting clean and malicious android applications. *Future Generation Computer Systems*, 36:122–132, 2014.
31. Xing Liu and Jiqiang Liu. A two-layered permission-based android malware detection scheme. In *Mobile cloud computing, services, and engineering (mobilecloud), 2014 2nd ieee international conference on*, pages 142–148. IEEE, 2014.
32. Shina Sheen and Anitha Ramalingam. Malware detection in android files based on multiple levels of learning and diverse data sources. In *Proceedings of the Third International Symposium on Women in Computing and Informatics*, pages 553–559. ACM, 2015.
33. Yury Zhauniarovich, Maqsood Ahmad, Olga Gadyatskaya, Bruno Crispo, and Fabio Massacci. Stadyna: addressing the problem of dynamic code updates in the security analysis of android applications. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, pages 37–48. ACM, 2015.
34. Aiman A Abu Samra, Kangbin Yim, and Osama A Ghanem. Analysis of clustering technique in android malware detection. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2013 Seventh International Conference on*, pages 729–733. IEEE, 2013.
35. Yang Wang, Jun Zheng, Chen Sun, and Srinivas Mukkamala. Quantitative security risk assessment of android permissions and applications. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 226–241. Springer, 2013.
36. Asaf Shabtai, Yuval Fledel, and Yuval Elovici. Automated static code analysis for classifying android applications using machine learning. In *Computational Intelligence and Security (CIS), 2010 International Conference on*, pages 329–333. IEEE, 2010.
37. Apktool. (Visited April 2017) [Online]. Available: https://ibotpeaches.github.io/Apktool/.
38. J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.
39. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.
40. Xindong Wu, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, S Yu Philip, et al. Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1):1–37, 2008.

41. Android Market API. (Visited April 2017) [Online]. Available: http://code.google.com/p/android-market-api.
42. Virus Total. (Visited April 2017) [Online]. Available: https://www.virustotal.com/.
43. Yajin Zhou and Xuxian Jiang. Dissecting Android Malware: Characterization and Evolution. *2012 IEEE Symposium on Security and Privacy*, (4):95–109, May 2012.