UpDroid: Updated Android Malware and Its Familial Classification

Kursat Aktas[1] and Sevil Sen[1] ✉

WISE Lab, Department of Computer Engineering
Hacettepe University, Turkey
kurtas.ce@gmail.com, ssen@cs.hacettepe.edu.tr

Abstract. Android is the platform most targeted by attackers. While security solutions have improved against such attacks on one side, attackers introduce new variants of existing malware by employing new strategies to evade them on another side. One of the most effective evasion techniques widely used is updating malicious code at runtime. In this study, an up-to-date dataset of such update attacks called UpDroid is introduced and then analyzed. This dataset consists of 2,479 samples belonging to 21 malware families, of which most have been discovered in just the last few years. While this dataset gives an overview of recent malware, it will also be useful for researchers working on dynamic analysis. Furthermore, in this study, a new classification algorithm based on both static and dynamic features is introduced in order to group such malware into families.

Keywords: Android, mobile malware dataset, update attacks, dynamic code loading, family classification, static analysis, dynamic analysis

1 Introduction

Android is still the platform most targeted by attackers [30]. According to a recent Av-test report [7], the number of malicious programs targeting Android has more than doubled in the last year. Mobile malware could damage end-users through different aspects such as stealing banking information, gaining root access and thereby corrupting the victim's device. However, the primary motivation of attackers is still driven by illicit financial gain [30]. Even Android has modified its architecture to improve security, but that is only beneficial to users who download the latest version of Android, which is rarely the case [30].

In the last few years, there has been significant growth in the number of new Android mobile malware variants, but a drop in the number of new Android mobile malware families [29][30]. Therefore, attackers are applying advanced evasion techniques to existing malware. Updating application at runtime is one of the most effective evasion strategies reported in the literature [8][25]. Since most commercial anti-virus solutions are based on static analysis, they prove largely ineffective against such update attacks. On the other hand, how to detect such

attacks and how to trigger them at runtime is an area needing further investigation. In order to accelerate studies in detecting update attacks and developing dynamic analysis-based solutions, a dataset of update attacks is introduced in this study. Such a dataset would prove useful for studies working on malware that cannot be effectively detected by static analysis-based techniques. Researchers working on input generation tools, fuzzing, and dynamic analysis could therefore refer to this dataset. Even though the current mobile malware datasets consist of some update attacks [4][34][37], this study introduces a dataset called UpDroid which consists entirely of update attacks. The study also presents analysis of the dataset, which is generally made up of different malware families than those found in other datasets [4][34][37]. Even though UpDroid contains few families that are common to the biggest recent malware dataset, AMD [34], it contains different samples of those families. Furthermore most of the new families included in UpDroid were released during or since 2015 (57.1%). Hence, this dataset is also intended to be useful for studies working on up-to-date mobile malware issues. The UpDroid dataset consists of 21 families and 2,479 samples. More than half of these families (12 out of 21) were discovered during or since 2015, while the remainder (9 out of 21) were discovered before 2015, and half of those (4 families) have new variants discovered either in 2015 or since then.

During construction of the UpDroid dataset, some difficulties were faced with familial classification of some malware, especially when most AV solutions could not reach a decision on a single family. It is known that familial classification of existing anti-virus solutions can be unreliable [14][15][16]. Therefore, in the current study, a new mobile malware family classification system is introduced based on both static and dynamic application features. With the proliferation of obfuscated and evasive malware, it is believed that using dynamic features has become inevitable for the correct classification of malware families.

Malware familial classification has become significantly important with the increased number of mobile malware variants seen in recent years. If the family of a detected malware is known, specific steps can be taken to decrease or reverse the damage caused by the malware. Furthermore, it helps to decrease the number of samples that malware analysts need to analyze. Automatic categorization of a harmful application into its family provides security professionals with an idea about the malware before carrying out the necessary manual analysis, and thereby, minimizes analysis time. To the best of our knowledge, only one recent study called Ec2 [9] has proposed malware family classification by applying hybrid features. The current study shows that the solution introduced achieves a better rate of accuracy than the results published for Ec2, by using fewer features.

2 UpDroid Dataset

This study introduces the UpDroid dataset to the research community [1]. The dataset consists of malicious applications using updating techniques in order to

[1] https://wise.cs.hacettepe.edu.tr/projects/updroid/dataset/

evade detection. An update attack typically does not contain any malicious code at the installation stage, waiting instead to add its malicious payload at runtime. The loading of a malicious payload could happen at the start of the application or it could use other triggering mechanisms such as event-based or, time-based [8]. For attackers, there are different ways to load their malicious code. One of the most used techniques is loading Java classes at runtime via ClassLoader objects. In such a case, the loaded code can be retrieved from the apk file or from a remote server at runtime. Another method is loading native code by using JNI (Java Native Interface). Android enforces applications to use defined APIs for loading native code. However, this loaded native code can also load and execute other native code without using this API. In addition, the most recently loaded native code can be stored as data and then be interpreted as code after loading. Because of these reasons, providing security against update attacks using this technique is more difficult than the class loading technique [24]. The last technique acquires the malicious payload by using the Package Manager Service, which manages the installation and deletion of applications in Android. Through this method, the application requires the user's confirmation in order to use the Package Manager. Therefore, it needs to use phishing techniques in order to persuade the user of its authenticity. The attacker then downloads and installs the actual malware after gaining the root privilege. Because of the technique employed, these types of malware are known as *Dropper* or *Downloader*.

The construction of the UpDroid dataset was carried out in three steps, as shown in Figure 1. Each step is explained in detail in the subsequent sections.

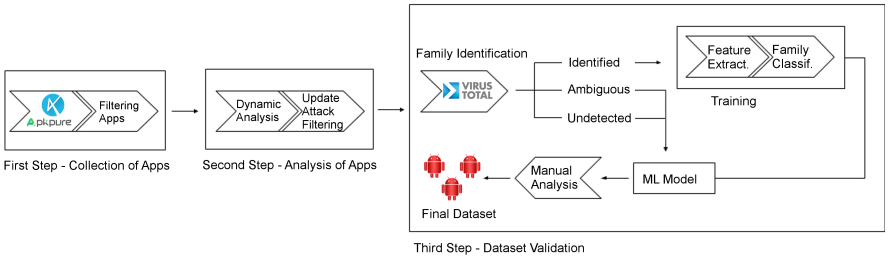


Fig. 1: Construction Steps of UpDroid

2.1 Collection of Apps

In this phase, Android applications were collected from the Koodous platform [19] and the ApkPure market store [3]. Koodous is a web platform for malware analysts which has built-in analysis tools such as DroidBox [2]. Apkpure is an unofficial application market. In order to obtain the dataset, three filtering mechanisms were applied to the downloaded samples from Koodous. The first filter downloads applications from the most recently uploaded to Koodous. The

primary reason is being to collect up-to-date malware. The second filter selects applications not detected as malicious by other analysts, since the aim is to add novel update attacks besides those already known to the dataset. For the last filter, output of the built-in Dropbox tool in Koodous was employed, which checks for applications with at least one loading activity using *DexClassLoader*. The most popular applications from each category are downloaded from Apkpure. As a result, 11,490 applications were obtained from Koodous, together with 6,299 applications from Apkpure.

2.2 Analysis of Apps

In this phase, the applications were run on an emulator for dynamic analysis. In order to do that, each application was run for 15 minutes and their DroidBox (4.1.2) outputs collected. All the file accesses that made by applications and all the network traffic that applications generate was logged for further analysis.

Malicious applications can be triggered in many ways. For example, the malicious payload could be loaded after a button is pressed, or after a certain period of time has passed [8]. Triggering techniques could vary by malware family. System events are one of the most used triggering techniques among malware. During the analysis, it was observed that *BroadcastReceiver* registered during runtime was one of the triggering mechanisms that update attacks apply. For example, an application which loads malicious code at runtime may use the *PackageAdded* receiver to ensure that the malicious package has been added. In this scenario, an attacker could use the *registerReceiver* function in order to add this receiver and bypass static analysis. Since DroidBox does not record registered receivers at runtime, the Android image was recreated by adding this log to the *registerReceiver* function on the Android framework. The other integration to DroidBox is the Monkey tool, which is used for random input generation of an application. Surprisingly, it is shown that the tools based on random exploration strategies such as Monkey obtained higher code coverage than more sophisticated strategies implemented by other tools [10]. Therefore, Monkey was chosen to be used in the current study.

Three filtering mechanisms were employed to the DroidBox outputs to find potential update attacks. All applications collected from Koodous and ApkPure in the first phase were sent to those filtering mechanisms in order to find malware. Since most of the samples were collected from Koodous, they were already expected to be malicious, but here the aim was to process update variants. In the first filter, the relationship betweeen dynamic code loading and data leakage has examined. A malicious application typically uses some personal sensitive data such as IMEI, IMSI, or phone number in order to identify the victim's devices. This data is generally leaked through the file system or over the network. This filtering mechanism basically checks whether or not the app has both dynamic loading and data leaking activities. If the code loading activity happens before the data leakage, then the application is considered as a possible candidate for update attack. The second filtering mechanism checks for an opened connection

from the app to a malicious server after dynamic code loading. For this purpose, all IP addresses fetched from DroidBox's output are separated into two groups (malicious and benign addresses) by using more than 200 blacklists [1]. Again, if the class loading activity happens before the connection to a malicious server, then the application is labeled as a possible update attack. If the reverse order happens, it is also considered as malicious, but since it is not an update attack, it is discarded. In the third filter, both static and dynamic analysis are employed to explore the relationship between native code loading, and both sensitive data leakage and malicious server connection. If the app's source code has *System.loadLibrary* or *System.load* functions to import native codes, and it performs one of the two malicious activities at runtime, then the application is labeled as a possible update attack.

2.3 Dataset Validation

The final step was validation of the constructed dataset. Here, all potential candidate update attacks are sent to the VirusTotal [32] at first. If the application is found to be malicious by more than 20 AVs and its dominant label given by AVs belonging to an update attack family, it is kept in the dataset. This step could have been directly applied to the collected applications in the first phase, but the second phase was still carried out for the possibility of exploring novel update attacks. Furthermore, results showed the filtering mechanisms to be sufficiently effective as 82.66% of potential applications sent to the VirusTotal [32] were confirmed as update attacks. Others undetected by VirusTotal [32] do not necessarily mean that they are not update attacks. Therefore, they are sent for the manual analysis as a possible new update attack. Manual analysis found 10 new attacks. While these 10 samples are added to the dataset, others undetected by AVs are filtered out. Since these attacks showed adequate similarity to the dataset's existing families, they were not considered as novel update attacks. However, it should be noted that they are also not considered as malicious by AVs. Finally, only 7.1% of all collected samples escaped our filtering mechanisms. Among these applications, if there are samples belonging to update families in the dataset, these samples were also included to the dataset in order to increase the dataset size.

If the number of AVs that give the most used label was used for more than twice the number of AVs for the second most dominant label, then the most dominant label was assumed to be reliable. Otherwise, the malware family was assumed to be ambiguous. In this phase, 150 ambiguous malware was detected. Such ambiguous malware was sent to the family classification algorithm developed in this study. Those 150 samples whose families were identified by this algorithm were also included to the dataset. Details of the algorithm are explained in the subsequent section.

Table 1: The UpDroid Dataset Overview

Family	Sample	Discovered	Obfuscation	Category	Updating Tech.			Triggering Tech.		
					Code Load	Native Load	Dropper	Start	Event	Time
Asacub	66	2015	✓	Banking		✓		✓		
BankBot	33	2015/7	✓	Banking	✓		✓		✓	✓
Extension	9	2013		Generic		✓		✓		
FakeBank	8	2013/6	✓	Banking		✓			✓	
FakeFlash	6	2012/7	✓	Banking	✓		✓	✓		
FakeToken	12	2017	✓	Banking	✓		✓	✓		
Krep	5	2013		Generic	✓			✓		
Ksapp	2	2013		Generic		✓		✓		
Leech	7	2015	✓	Generic	✓				✓	✓
Lotoor	11	2010/3		Generic	✓		✓		✓	
Malap	193	2013	✓	Info Stealer	✓	✓			✓	
Marcher	30	2013/7	✓	Banking	✓					
Ogel	12	2015	✓	Generic		✓			✓	✓
Rootnik	41	2015/7	✓	Generic	✓		✓	✓		
Shedun	630	2015	✓	Generic	✓			✓		
SmsReg	291	2015	✓	Generic	✓		✓		✓	
SmsSpy	66	2014/6	✓	Banking	✓				✓	
Sprovider	5	2016	✓	Generic	✓				✓	
Tordow	11	2016	✓	Banking	✓		✓		✓	✓
Triada	1026	2016	✓	Generic	✓	✓			✓	✓
Ztorg	15	2015	✓	Generic	✓		✓	✓		

2.4 UpDroid Dataset Overview

As shown in Table 1, the UpDroid dataset has 21 malware families, and a total of 2,479 malware samples. In Table 1, for some families there are two discovery times because some new variants of these families were discovered after its first release into the wild. Overall, 51.7% of these families were collected during or after 2015. For observing the behaviors of malware families, a few samples from each family were manually analyzed. During this analysis, it was observed that new malware families are generally more sophisticated and complex than the previous ones. For example, all four families not employing obfuscation techniques were discovered prior to 2014; such as a sample of the Extension family that imports its malicious native code immediately after it started. The author of this malware is obviously not concerned with hiding the name of the native library to be loaded.

On the monetization side, the families are divided into three categories : banking, generic, and information stealing. Eight families belonging to the banking category try to steal victim's banking information. For example, Tordow targets banks in Russia and has root access gain capability, encrypting files and acting like ransomware besides the traditional banking malware [11]. It is observed that new malware families in Android platforms have more than one feature to harm victims such as Tordow. The generic category is used for families which have no specific target. They usually try to infect the device and gain root access. Upon obtaining the root privilege, they may, for example download other applications, or join a botnet. For instance, Rootnik removes its icon from the menubar immediately after installation on the device and tries to gain root privilege. After that, it tries to install aggressive advertisement applications. The last category belongs to malicious applications which steal sensitive information from victims.

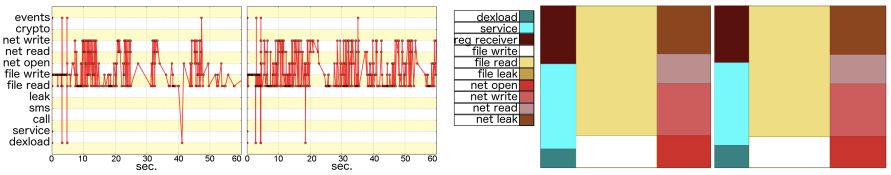
It is observed that 16 malware families use dynamic code loading, eight families use native code loading and seven families use dropper technique for updating itself. Another point is that most of the families use combinations of these techniques. For example, the 2015 variant of the BankBot family uses only dynamic code loading to import its malicious payload; whereas newer variants of the same family firstly download and install a new apk, and then this new apk uses dynamic code loading in order to import its malicious payload.

It is observed that nine malware families trigger their malicious code immediately after starting, whereas 11 families are event-triggered. These events could be inputs given by the user, or system events, etc. In addition, five families use time-based triggering [8]. Just like the updating techniques, some families use a combination of different triggering techniques. Interestingly, all these observed families were discovered during or after 2015, and use event-based and time-based triggering mechanisms together.

3 Family Classification

Malware samples belonging to a same family share some common features. Although the studies mainly focus on malware detection, malware family classification becomes more important each day due to increasing variations in each family. It is known that commercial anti-malware tools are not reliable in identifying the family [14][15][16]. A family classification algorithm was also needed for the construction of the UpDroid dataset created in the current study. When anti-malware solutions cannot agree on a family, the family classification algorithm introduced in this study was employed. While most family classification studies in the family classification rely on static features, they are seen as inadequate, since an attacker can easily change static features by using various methods like obfuscation, dynamic code loading, etc. Furthermore, some similarities between samples belonging to the same family were observed during this study's dynamic analysis (see Figure 2a and Figure 2b). Figure 2a shows the activity-time relations for two samples belonging to the SmsReg family. Both samples have similar sequences of activities, but at different times. Since they are randomly triggered, it is an expected result. But the density of activities carried out by each sample are very similar, as demonstrated in Figure 2b. Based on these observations, a family classification algorithm based on static and dynamic features was aimed to be investigated.

Besides the update attack families in the UpDroid dataset, new families are also included for familial classification. Here, only families identified by VirusTotal [32] are included in the dataset. Since AVs have different standards for naming malware and malware families, 342 family names were collected from the Internet [4][5][26][33] in order to extract family names from the outputs of AV. After the family names were extracted, they were checked similarly checked as per the AMD dataset's construction [34]. If the number of AVs that gave the most used label was used more than twice that of the number of AVs for the second most dominant label, then the most dominant label was assumed to be



(a) (b)

Fig. 2: (a)Activity-time Graphs, (b) Activity Density Graphs of Two Samples Belonging to the SmsReg Family

reliable. Otherwise, the malware was discarded. Since one of the main purposes of this study is to examine features for classifying new malware families, the families discovered in recent years were tried to be added to the dataset for the purposes of familial classification. Most of the families used in the dataset ($\approx$ 82%) were detected in the last five years. This dataset called *Last5Y* is mainly constructed from samples in Koodous. The MalGenome and Drebin datasets, which are mainly used for comparative purposes in the literature, were also evaluated in the results. The PRAGuard dataset [21] was also aimed to be evaluated in this study in order to assess the obfuscation resiliency of the proposed algorithm. However, the samples in this dataset could not be run on the emulators and, the authors of the dataset could not be reached on this matter.

3.1 The Method

For familial classification, features from both static and dynamic analysis were collected. Since the studies based on static analysis in the literature perform to a high level of accuracy, and dynamic features were observed to be similar for malware samples belonging to the same family in this study, feature selection is carried out both statically and dynamically. In the current study, the majority of static features were extracted from the Manifest file, as shown in Table 2. As pointed out in RevealDroid [16], permissions are very important for both malware detection and familial classification. Therefore, supported permissions by Android were used as boolean attributes. The count of custom permissions defined by the application were also added to the static features. Other static features are the number of activities, services and receivers given in the Manifest file and the size of the APK file. It was also observed that some update attack families (e.g., Shedun, Triada, etc.) define more activities, services and receivers in the Manifest file not used in the source code. Since these components are required for use by the downloaded code in the future, it should be defined as such in the Manifest file. Therefore, the existence of such extra components were also added as boolean values to the features list. To the best of the authors' knowledge, this study is the first to employ this feature.

For obtaining dynamic features, the applications were run on the DroidBox for a period of 15 minutes. Other than the total size of packets sent/received

through the network, all features demonstrated in Table 2 were extracted from DroidBox's output. The crypto activities were also collected. Here, the count of each activity (encryption, decryption, and key generation) was separately included in the features, which differed from the literature. All crypto algorithms (AES, RSA, etc.) supported by Android were used as boolean attributes. In order to observe data leakage, 23 sensitive tainted data (IMSI, IMEI, etc.) were monitored by DroidBox. All of them were included in the features list. DroidBox also logs the way the data is leaked (e.g. through sms, file, network). The data leakage numbers for each way was also used as features. Another feature is the number of registered receivers added to the Dropbox by the authors. It was observed that some applications read the */proc/meminfo*, */proc/cmdline*, */proc/event* directories which provide information about the system on which the application is run. Therefore, the number of read operations from these directories were also taken as a feature. Besides these directories, the number of read and write operations from the directory */data/data/appname*, in which an application has legal access to, was also added to the features. Finally, other outputs from the DroidBox were included. To the best of the authors' knowledge, this study is the first to use most of these dynamic features for familial classification. Some of these features are the number of registered receivers at runtime, the number of each data leakage, etc. As shown in Table 2, 175 features were collected in total.

Table 2: Features Used for Familial Classification

Feature Type	**Feature Explanation**	**Type**	**Count**
Static	Number of custom permissions	Numeric	1
Static	Existence of each permission	Boolean	102
Static	Number of activities	Numeric	1
Static	Number of services	Numeric	1
Static	Number of receivers	Numeric	1
Static	Existence of extra components	Boolean	3
Static	APK size	Numeric	1
Dynamic	Number of opened/closed network connections	Numeric	2
Dynamic	Number of unique network connections	Numeric	2
Dynamic	Total size of network packets	Numeric	1
Dynamic	Number of sent/received network packets	Numeric	2
Dynamic	Number of crypto activities	Numeric	3
Dynamic	Usage of crypto algorithms	Boolean	13
Dynamic	Existence of sensitive leaked data for each type	Boolean	23
Dynamic	Number of data leakage for each way	Numeric	3
Dynamic	Total data leakage	Numeric	1
Dynamic	Number of registered receivers at runtime	Numeric	1
Dynamic	Number of read/write operations from some directories	Numeric	6
Dynamic	Number of file accesses	Numeric	1
Dynamic	Number of file read/write operations	Numeric	2
Dynamic	Number of sent SMSs	Numeric	1
Dynamic	Number of started phone conversations	Numeric	1
Dynamic	Number of DexClassLoader usage	Numeric	1
Dynamic	Number of started services	Numeric	1
Dynamic	Number of crypto operations	Numeric	1

In this study, techniques based on machine learning were employed for familial classification. The Weka tool [17] was utilized in order to implement classification algorithms (J48, Random Forest, kNN) with their default parameters. A malware family which has very few samples in training could negatively affect the results. Therefore, families with fewer than 20 samples in both datasets were removed and 20 fold cross-validation employed. As a result, 25 families and 3,994 samples remained in the Last5Y dataset, and 24 families and 4,476 samples in the Drebin dataset.

3.2 Results

Results for the different classification algorithms are represented in Table 4. kNN outperforms other algorithms. The results are detailed in Table 4, where FP shows the weighted average of false positive rate of all families, and TP represents the weighted average of true positive rate of all families. As can be seen, the algorithms have high TP rates and very low FP rates for both datasets.

Table 3: Family Classification Results

Algorithm	Accuracy (%)			TP (%)			FP (%)		
	Last5Y	UpDroid	Drebin	Last5Y	UpDroid	Drebin	Last5Y	UpDroid	Drebin
kNN	92.41	96.37	96.85	92.4	96.4	96.8	0.5	0.2	0.3
Random Forest	91.08	96.2	95.87	91.1	96.2	95.9	0.5	0.4	0.6
J48	89.7	96.2	95.37	89.7	96.2	95.4	0.6	0.4	0.4

The proposed family classification approach called UpDroid was compared with Ec2 [9]. To the best of the authors' knowledge, Ec2 [9] is the closest work to the current study, since it is the only application using both static and dynamic features. Ec2 shows the performance of its algorithm on the Drebin dataset with families of more than 10 samples. It performs five fold cross-validation, and therefore, for the purposes of fair comparison, the same settings were applied in the current study. The comparison results are demonstrated with the same performance metrics as Ec2 uses (MiF-micro F-score and MiAUC-micro area under the curve) in Table 5. The results of the two common algorithms in each study are shown. While Ec2 gives the best results with Random Forest, the current study shows the best performance with kNN. However, overall, the current study's familial classification algorithm (kNN) shows better performance than Ec2 in each metric, especially in MiAUC, the metric which is indifferent to class imbalance. It shows also a high accuracy (95.05%) against the dataset containing small families that have at least two samples. Please note that while Ec2 finds the best parameters of classification algorithms by using hyper-parameter optimization, the default parameters of such algorithms are employed in the current study. Hence, no tuning is explicitly applied in order to outperform other studies. Furthermore, while EC2 employs 190 static and 2048 dynamic features, UpDroid

uses 175 features. This indicates the selected features' ability of distinguishing malware that is built by analyzing recent malware.

Table 4: Comparison with Ec2 [9]

Approach	Algorithm	MiF	MiAUC
Ec2	kNN	0.47	0.73
UpDroid	**kNN**	**0.96**	**0.98**
Ec2	Random Forest	0.95	0.97
UpDroid	Random Forest	0.94	0.99

Since most of the family classification studies use the MalGenome dataset for evaluation, the performance of the proposed algorithm is also assessed on this dataset. Since kNN produces the highest level of accuracy, it was elected to be employed across all other relevant experiments of this study. For the purposes of fair comparison, the same settings that FalDroid [15] used were applied in the current study. All families having more than one sample is taken into account, and 10-fold cross validation is applied. Table 6 compares the results with some known static analysis-based classification algorithms in the literature. UpDroid is clearly one of the best family classifiers. UpDroid is also effective on differentiating families with few samples. FalDroid [15] and RevealDroid [16] are the most recent works. While FalDroid shows comparable results with UpDroid, UpDroid shows better performance than RevealDroid [16] considerably. DroidSieve [27] performs slightly better on the MalGenome dataset (97.79%). Since DroidSieve explores the use of obfuscation-invariant features for both malware detection and family identification, it is quite effective against obfuscated malware. However, it should be noted that it uses a different experimental setting (66% split for training) than the current study. Since the current study mainly focuses on dynamic features for detecting obfuscated malwares, such static obfuscation-invariant features could be considered to be added to the hybrid approach in the future.

Table 5: Comparison with Static Analysis-Based Approaches

Approach	Accuracy
DenDroid [28], 2014	94.2%
DroidSIFT [36], 2014	93.0%
Droidlegacy [14], 2014	92.9%
DroidSieve [27], 2017	97.79%
FalDroid [15], 2018	97.2%
RevealDroid [16], 2018	95.0%
UpDroid	97.32%

DroidScribe is the only familial classification work that is based on dynamic features. Even though it improves the classification accuracy from 84% to 94% by

using Conformal Prediction, UpDroid still achieves a much better performance (96.85%) on the same dataset, Drebin with the same setting.

False positive rates are quite low for all families. The highest false positive rate is 0.7% for Adrd in Drebin and 0.9% for SmsReg in Last5Y. On the other hand, except for the Boxer, SmsReg, Gappusin and LinuxLootor families, all families show more than 90% true positive rate in Drebin. The same families also decreased detection rates in other studies [4][9], due to being difficult to trigger, and hence to detect. In contrast to Drebin, new malicious software seems more sophisticated in order to evade security mechanisms. Nearly half of the families has true positive rate under 90%. Especially the Youmi, Dowgin, and Kuguo families decreased the overall results. As shown in the confusion matrix in Figure 3, these families are confused with each other. After analysis of these families, many similarities among them are observed. All these families are obfuscated aggressive advertisement malware, using the same installation strategy and the same triggering mechanism to, steal the device information [34].

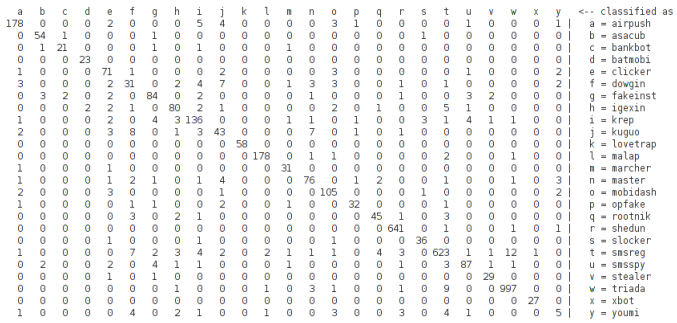


Fig. 3: Confusion Matrix for the Last5Y Dataset

4 Related Work

There are typically two datasets used for comparison in research papers : MalGenome [37] and Drebin [4]. The MalGenome dataset was the first mobile malware dataset introduced in 2011. It consists of 1,260 malware from 49 different malware families. Drebin [4] is a larger dataset which was introduced in 2014, and which also contains malware from the MalGenome dataset. A dataset of obfuscated malware called PRAGuard dataset [21] was then released in 2015; having collected malware from the MalGenome [37] and the Contagio [12] datasets [12], and then applied different obfuscation techniques to them. The most recent dataset introduced to the mobile security community is AMD [34]. It contains 24,553 samples from 71 malware families collected between 2010 and 2016. Besides these datasets, some researchers also share datasets used in their studies

(e.g. [16][20]). However, to the best of the authors' knowledge, there is no public dataset to be found in the literature that especially focuses on update attacks.

Although there have been many studies for Android malware detection, they generally do not perform malware family classification. The first work on automated familial classification is Droidlegacy [14]. The study classified piggybacked applications based on the assumption that the most common code in a family would be malicious and the Android API calls used by such malicious code are used as the signature of a family. Although it performed with 98% accuracy on family classification for a set of 11 families, the basic assumption lead to misclassification of some applications, since they shared the code of a supporting library (e.g. advertisement libraries) as the most common code. Other early studies on mobile malware family classification were Dendroid [28], Droidminer [35], and DroidSIFT [36]. Dendroid [28] proposed family classification based on statistical analysis of an application's code structure. While Droidminer [35] constructed the behaviour graph of an application, DroidSIFT [36] was based on the similarity of API dependency graphs of applications. A recent work also proposed a familial classification system based on call graph similarity [15], in which, the sensitive API call-related graph (SARG) of an application was extracted, and which reduced the size of the function call graph of an application by approximately 72%. In order to calculate the similarity of SARG graphs, a new method based on TF-IDF was proposed. Representative samples of each family were determined based on this similarity metric, which greatly reduced the workload for malware analysts. Although the study achieved a higher level of accuracy (95% for Drebin dataset) compared to other approaches based on static analysis, it was not resilient to control flow obfuscation techniques. Another recent study based on code similarity at the methods' level also suffered from code obfuscation techniques [22].

Recently, two further studies were proposed in order to detect and classifiy obfuscated malwares. RevealDroid [16] takes into account features which could help detect obfuscated malwares such as reflection-based and native-code features. The result showed RevealDroid to be a lightweight and obfuscation-resilient approach compared to others, namely MUDFlow [6], Drebin [4], and Dendroid [28]. DroidSieve also explores the features for both detecting obfuscated and non-obfuscated malwares, and introduced a set of novel features. It was shown that the high-ranked features of malware classification for plain and obfuscated malwares showed a degree of similarity. For example, it was shown that permissions and used-permissions play an important role in detecting both plain and obfuscated malware. Another study which employed only requested permissions in the Manifest file for family classification [23] also supported this result.

To the best of the current study's authors knowledge, there has only been one study called Droidscribe [13] that is based on the dynamic features of malware for family classification. In the study, features related to the following groups were extracted by using CopperDroid [31]: network access, file access, binder method, and execute file. By applying SVM multi-class classification on these features, the

proposed approach achieved 84% accuracy on the Drebin dataset [4]. Although the results were quite low when compared to static analysis-based approaches, they increased up to a level of 94% accuracy by Conformal Prediction (CP), which is a computationally expensive algorithm. Therefore, it is only applied to malware for which SVM does not meet the desired classification quality. On the other hand, CP returns a possible list of classes to which malware belongs. Therefore, additional analysis is needed in order to select the right class, even after applying CP.

Even though there are some research on investigating hybrid features of malware against Windows systems [18], there has only been one hybrid study on mobile platform, called Ec2 [9], which is a very recent addition to the literature. The most important contribution of this work was the classification of small malware families with less than 10 samples. In order to achieve that, an ensemble approach which combines clustering and classification techniques in a systematic way was proposed. Ec2 showed good results both on the Drebin and Koodous datasets. It was also shown to perform better than Droidlegacy [14]. In the current study, an effective hybrid-based family classification algorithm using much fewer features is proposed.

5 Conclusion

A new dataset called UpDroid is introduced in this current study. Although there are a few existing mobile malware datasets to be found in the literature, UpDroid differs by only focusing on update attacks. Since updating techniques are one of the most commonly used evasion strategies used by attackers, solutions need to be developed against such attacks. It is believed that this dataset will accelerate studies working on malware which cannot be effectively handled by static analysis alone. This dataset consists of 2,479 samples belonging to 21 malware families; most of which were discovered in the last few years. The analysis of this dataset both provides information about the update attacks and recently introduced malware.

Since update attacks download their malicious code at runtime, detection and familial classification based static analysis techniques are largely ineffective against such attacks. As also encountered while constructing the UpDroid dataset in this study, these attacks cannot be correctly grouped into families by using the anti-virus solutions available on the market. While most of the familial classification algorithms in the literature are based on static features, they are largely ineffective against update attacks or malware using obfuscation techniques. As recent malware usually employs such evasion techniques [34], an obsufcation-resilient family classification algorithm is needed. Therefore, this study introduces a new familial classification algorithm based on both static and dynamic features. This algorithm shows better performance than Ec2 [9], which is the first and only other familial classification algorithm to employ hybrid features, by using much fewer discriminating features. The introduced algorithm

achieves a high degree of accuracy and a low false positive rate on both the recent malware and the samples in the Drebin dataset.

6 Acknowledgment

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. blcheck: Test a mail servers against black lists. (March 2018), https://github.com/darko-poljak/blcheck
2. Droidbox: Dynamic analysis of android apps (March 2018), https://github.com/pjlantz/droidbox
3. Apkpure: Android market place (March 2018), https://apkpure.com/
4. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K.: Drebin: Effective and explainable detection of android malware in your pocket. In: Proceedings of the Network and Distributed System Security (NDSS) Symposium (2014)
5. Ashishb: android-malware (March 2018), https://github.com/ashishb/android-malware
6. Avdiienko, V., Kuznetsov, K., Gorla, A., Zeller, A., Arzt, S., Rasthofer, S., Bodden, E.: Mining apps for abnormal usage of sensitive data. In: Proceedings of the 37th International Conference on Software Engineering-Volume 1. pp. 426–436. IEEE Press (2015)
7. AVTEST: Security report 2016/2017 (2017), https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2016-2017.pdf
8. Aysan, A.I., Sen, S.: 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. pp. 181–186. IEEE (2015)
9. Chakraborty, T., Pierazzi, F., Subrahmanian, V.: Ec2: Ensemble clustering and classification for predicting android malware families. IEEE Transactions on Dependable and Secure Computing (2017)
10. Choudhary, S.R., Gorla, A., Orso, A.: Automated test input generation for android: Are we there yet?(e). In: Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on. pp. 429–440. IEEE (2015)
11. Comodo: Comodo threat research labs warns android users of tordow v2.0 outbreak (March 2018), https://blog.comodo.com/comodo-news/comodo-warns-android-users-of-tordow-v2-0-outbreak/
12. Contagio: contagio (March 2016), http://contagiodump.blogspot.com.tr/
13. 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: Security and Privacy Workshops (SPW), 2016 IEEE. pp. 252–261. IEEE (2016)
14. Deshotels, L., Notani, V., Lakhotia, A.: Droidlegacy: Automated familial classification of android malware. In: Proceedings of ACM SIGPLAN on program protection and reverse engineering workshop 2014. p. 3. ACM (2014)
15. Fan, M., Liu, J., Luo, X., Chen, K., Tian, Z., Zheng, Q., Liu, T.: Android malware familial classification and representative sample selection via frequent subgraph analysis. IEEE Transactions on Information Forensics and Security (2018)

16. Garcia, J., Hammad, M., Malek, S.: Lightweight, obfuscation-resilient detection and family identification of android malware. ACM Transactions on Software Engineering and Methodology (TOSEM) 26(3), 11 (2018)
17. Hall, M.e.a.: " the weka data mining software: an update. In: SIGKDD Explor. 11. pp. 10–18. ACM (2009)
18. Islam, R., Tian, R., Batten, L.M., Versteeg, S.: Classification of malware based on integrated static and dynamic features. Journal of Network and Computer Applications 36(2), 646–656 (2013)
19. Koodous: Online malware analysis platform (March 2018), https://koodous.com/
20. Lindorfer, M., Neugschwandtner, M., Weichselbaum, L., Fratantonio, Y., Van Der Veen, V., Platzer, C.: Andrubis–1,000,000 apps later: A view on current android malware behaviors. In: Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS), 2014 Third International Workshop on. pp. 3–17. IEEE (2014)
21. Maiorca, D., Ariu, D., Corona, I., Aresu, M., Giacinto, G.: Stealth attacks: An extended insight into the obfuscation effects on android malware. Computers & Security 51, 16–31 (2015)
22. Marastoni, N., Continella, A., Quarta, D., Zanero, S., Preda, M.D.: Groupdroid: Automatically grouping mobile malware by extracting code similarities. In: Proceedings of the 7th Software Security, Protection, and Reverse Engineering/Software Security and Protection Workshop. p. 1. ACM (2017)
23. Ping, M., Alsulami, B., Mancoridis, S.: On the effectiveness of application characteristics in the automatic classification of malware on smartphones. In: Malicious and Unwanted Software (MALWARE), 2016 11th International Conference on. pp. 1–8. IEEE (2016)
24. Poeplau, S., Fratantonio, Y., Bianchi, A., Kruegel, C., Vigna, G.: Execute this! analyzing unsafe and malicious dynamic code loading in android applications. In: NDSS. vol. 14, pp. 23–26 (2014)
25. Qu, Z., Alam, S., Chen, Y., Zhou, X., Hong, W., Riley, R.: Dydroid: Measuring dynamic code loading and its security implications in android applications. In: Dependable Systems and Networks (DSN), 2017 47th Annual IEEE/IFIP International Conference on. pp. 415–426. IEEE (2017)
26. Spreitzenbarth: Current android malware (March 2018), https://forensics.spreitzenbarth.de/android-malware/
27. Suarez-Tangil, G., Dash, S.K., Ahmadi, M., Kinder, J., Giacinto, G., Cavallaro, L.: Droidsieve: Fast and accurate classification of obfuscated android malware. In: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy. pp. 309–320. ACM (2017)
28. 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 41(4), 1104–1117 (2014)
29. Symantec: Internet security threat report (April 2016), https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf.
30. Symantec: Internet security threat report, volume 22 (April 2017), https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf
31. Tam, K., Khan, S.J., Fattori, A., Cavallaro, L.: Copperdroid: Automatic reconstruction of android malware behaviors. In: NDSS (2015)
32. VirusTotal: Virustotal (March 2018), https://www.virustotal.com

33. Website, A.: Android malware behaviors (March 2018), http://amd.arguslab.org/behaviors
34. Wei, F., Li, Y., Roy, S., Ou, X., Zhou, W.: Deep ground truth analysis of current android malware. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'17). pp. 252–276. Springer, Bonn, Germany (2017)
35. Yang, C., Xu, Z., Gu, G., Yegneswaran, V., Porras, P.: Droidminer: Automated mining and characterization of fine-grained malicious behaviors in android applications. In: European symposium on research in computer security. pp. 163–182. Springer (2014)
36. Zhang, M., Duan, Y., Yin, H., Zhao, Z.: Semantics-aware android malware classification using weighted contextual api dependency graphs. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. pp. 1105–1116. ACM (2014)
37. Zhou, Y., Jiang, X.: Dissecting android malware: Characterization and evolution. In: Security and Privacy (SP), 2012 IEEE Symposium on. pp. 95–109. IEEE (2012)