Family Matters: On the Investigation of [Malicious] Mobile Apps Clustering

Thalita Scharr Rodrigues Pimenta[1(✉)], Rafael Duarte Coelho dos Santos[2], and André Grégio[3]

[1] Federal Institute of Paraná, Irati, Brazil
thalita.pimenta@ifpr.edu.br
[2] INPE, São José dos Campos, SP, Brazil
rafael.santos@inpe.br
[3] Federal University of Paraná, Curitiba, Brazil
gregio@inf.ufpr.br

Abstract. As in the classification of biological entities, malicious software may be grouped into families according to their features and similarity levels. Lineage identification techniques can speed up the mitigation of malware attacks and the development of antimalware solutions by aiding in the discovery of previously unknown samples. The goal of this work is to investigate how the use of hierarchical clustering on malware statically extracted features can help on explaining the distribution of applications into specific groups. To do so, we collected 76 samples of several versions from popular, legitimate mobile applications and 111 malicious applications from 11 well-known scareware families, produced their dendograms, and discussed the outcomes. Our results show that the proposed apporach is promising for the verification of relationships found between samples and their attributes.

Keywords: Clustering · Mobile malware · Lineage

1 Introduction

The development of signatures/heuristics to detect malicious mobile applications is a complex and time-consuming task, due to the large number of daily releases in app stores. According to McAfee Mobile Threat Report, more than 1.5 million variants of malicious code in mobile applications were detected in the first four months of 2019 [29]. Applying different types of analysis allows the classification of malware in groups or families and, consequently, the identification of suspicious behaviors and/or attributes extracted from mobile applications. Antiviruses (AVs) for the Android platform are based on package databases, file location in the operating system, and information from mobile applications [9]. Non-malicious and malicious applications use an extensive quantity of attributes like permissions, intent filters, external classes, and components. AV scanners check the packages and their locations before allowing the application installation. However, files added after installation may be unable to be monitored

© Springer Nature Switzerland AG 2021
O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12951, pp. 79–94, 2021.
https://doi.org/10.1007/978-3-030-86970-0_7

by AVs [9]. Therefore, external classes and the dynamic download of additional components may cause installed malware to escalate privileges.

Some types of malware, like Ransomware and Scareware call external components and classes to manipulate users. Scareware, which can be classified as a type of ransomware, takes advantage of violations related to the user experience [7]. Scareware tricks users with false applications that leverage security warnings on their devices. These fake treats intend to sell useless services, allowing the theft of bank information, such as the victim's credit card details. Installing these services makes it easier to prepare the device for Ransomware-type attacks, which encrypt files and require a ransom to be paid, usually with cryptocurrencies. For instance, in 2020, the 'COVID-2019 Scareware' used a fake data encryption threat and demanded ransom to terrify its victims [23]. Although not as harmful as traditional crypto-ransomware, this type of malware support scams and information theft, since this malicious code presents dialog boxes demanding ransom, without having genuine encryption features. During the COVID-19 pandemic, the terms related to the disease have been used for disseminating malware and fraudulent pages, as happened in Canada [5]. According to Norton UK [32], criminals have been using search engine optimization poisoning, a type of attack that associates malicious web pages with popular search terms. The malicious page is listed in the search results, looking legitimate to the end-user.

Due to the importance of mitigating this type of threat to the current cybercrime scenario, it is beneficial to develop alternative security solutions, considering device hardware limitations, operating system peculiarities, and end-user expertise. Cloud database-based solutions with attribute lists assigned to known families can be queried and updated by users, particularly enabled by pop-up windows and other components used by these plagues.

Concerning the window for identifying new threats that can last for months [3], we investigate the application of unique attributes and hierarchical clustering to recognize unknown variants of known malware families. In the literature, it is common to encounter hierarchical clustering analysis for malware classification and goodware. However, it is rare to find attempts to explain the divisions between families in the resulting clusters. Another little-explored sub-area is the investigation of the diversity in characteristics and behaviors found in samples from the same family. These variances can result in other branches or even other labels, depending on the antivirus consulted.

To be able to investigate the attributes usually used in mobile malware, it is interesting to compare with sets of elements of non-malicious applications. Thus, in conjunction with the malicious samples, versions of ten benign applications were analyzed. The goodware samples are from known apps present in the most downloaded general apps ranking from the Android system app store. In our experiments, a dataset of 111 samples of 11 Scareware families was used. The contributions of this paper are twofold: (i) we show and discuss the reasons behind the distribution of malicious and non-malicious app among their corresponding clusters, concluding that what differentiates them is the use of external

and unverified sources, peculiar strings, and uncommon URLs, also often called to the same vulnerable attributes, principally permissions; (ii) we analyze which of the extracted features unique to Android malware families aid in the identification of Scareware type, aiming to foster a base of external code lists and critical strings, when found concomitantly with distinct attributes, for dynamic monitoring of application operations.

The rest of the paper is organized as follows. In Sect. 2, we discuss the closest available work and position ours, and a short description for Android features and clustering is given. In Sect. 3, the proposed classification method is presented. The results discussions are explained in Sect. 4. Conclusions and future directions for research are shown in Sect. 5.

2 Background and Related Work

In the context of malware analysis, variants are examples of a group or family, which share similar properties, behaviors, and structures. Features are software elements used to classify code as malicious or not.

According to the literature, there is a time window for new variants to be noticed by security analysts and/or end-users. For Android malware analysis, researches indicating that one can spend about three months to detect new malware [3].

The malware classification problem has cumulative complexity due to the increasing number of characteristics necessary to represent the application's behavior exhaustively [19]. Certain permissions and services can be used by normal or malicious applications. However, some attributes are more frequent in illegal activities. For example, numerous event logs related to boot, SMS messages, phone call logs, and package management features are related to financial malware [1].

The properties present in the application can be extracted from several methods, mainly through static (without code execution), dynamic (with code execution), or hybrid analysis techniques, combining static and dynamic methods [12]. In addition to the extraction methods, several forms of representation of the collected features are found in the literature. Some popular techniques are listed below: *N-grams* [14], *opcodes* [27], sequences of function calls [30], flow control graphs [2], data flow graphs [33], dangerous API calls [31] and network communication data [24].

It is noteworthy that depending on the sample's complexity, it is necessary to investigate the extraction methods for more appropriate characteristics, directly influencing the quality of the results obtained after analysis.

2.1 Android Architecture and Analyzed Components

The AndroidManifest.xml file displays statements about the operating system resources and hardware assets used in the application. As enabling usage permissions and functionality can be dangerous, this file is employed to detect anoma-

lous and unsafe behavior [10]. About the Android architecture, the values of the following components are extracted:

- **Intents:** These components provide data about operations and behaviors that can be activated by the application. Examples of resources for sharing data and services between applications include: *Activities*, *Intents*, *Broadcast Receivers* and *Services* [18].
- **Permissions:** These are authorizations to access the hardware resources and system functionalities. For example, some applications store user location data via GPS. Others may ask for information about the contact list. Thus, the control of relevant parts for security and privacy is carried out by the permission system, when the application is installed [11].
- **External Classes:** Applications for the Android platform, malicious or not, run external code. Still, these mechanisms can result in security issues, such as evasion of malware detectors and Code Injection attacks [26]. Consequently, the investigation of methods of external classes called by the application allows the verification of unsafe behaviors.

2.2 Clustering and Similarity

Due to the large number of variants that have been created automatically, which bypasses detection mechanisms of patterns and signatures, manual analysis has become a significant bottleneck in the control of virtual pests [4].

Several techniques (e.g., phylogeny, clustering) have been applied over the years to identify whether a mobile application is malicious or not. Clustering is an unsupervised learning technique for finding patterns in data. Given the difficulty of analyzing all the possibilities of groups in a large amount of data, several techniques were created to assist the formation of the clusters.

The use of these techniques intends to identify unknown malware mainly by calculating the similarity of a sample with previously grouped, already known families of malicious apps. While the similarity between two strongly related objects has a value of 1, the distance has a value of zero and vice versa [22]. Examples of popular indices include the Jaccard Index and Euclidean distance. Complementary equations and information can be found in the study of [6].

Grouping methods known as Hierarchical Clustering Algorithms create the hierarchy of relationships between objects. The central aspect of these algorithms is the pair selection based on a connection measure between the elements. The linkage measure uses similarity or distance indices to cluster similar objects. The most common forms of bonding are Complete Linkage, Simple Linkage, Average Linkage, Centroid Methods, Median Method, and Ward Method. Regarding the quality analysis of the generated clusters, internal and external indexes can be applied. External indexes assess inter-group variations and distances, while internal indexes assess intra-group variations. Examples of popular indices include Purity, Entropy, Precision, Recall and *F-Score* [28].

The quality of the clustering methods results depends on the chosen algorithm, parameters, and similarity measures [20].

2.3 Related Work

In the literature, several papers applied hierarchical clustering for Android malware detection and/or classification. However, we selected some related works focused on hierarchical clustering and relationships between families. Zhou et al. carried out characterization and evolution studies of more than 1200 samples from 49 different families, collected between 2010 and 2011. The results showed that 86% samples collected are legitimate applications packaged with malicious payloads [34].

Pfeffer et al. presented a system for analyzing malware and its attributes using genetic information, called MAAGI. From a bag-of-words model, three types of malware characteristics are used: two strings and one type of N-gram with a call procedure graph. The first strings are extracted from the binaries and the used libraries. In the clustering part, the Louvain method, based on the graphs grouping, was used together with K-means. Also, they present a method for extracting the lineage of malware using a probabilistic model. Thus, as a result, the authors believe that the system is the first to integrate clustering, lineage, and prediction of malware and families [25].

Hsiao et al. highlight that several threads are used to perform intrinsically complex tasks in Androids applications. Thus, it becomes difficult to analyze malicious applications without prior knowledge of their structures. Based on behavior analysis, the authors propose an investigation scheme, which involves phylogenetic trees, UPGMA, significant main components, and a matrix of points. For the experiments, more than 980 samples from different families were studied, making it possible to detect which code was shared between "sibling" applications and which was inherited from common ancestors [13].

In [17], the authors presented an Android malware clustering technique by mining malicious payloads. The proposed solution checks whether the applications share the same version of malicious payload. The authors created a method to remove code from legitimate Android application libraries while still keeping malicious payloads, even if they are injected under the name of popular libraries. In experiments carried out, the accuracy rate was 0.90 and the recall was 0.75 for the Android Gnome malware set.

Time is essential when dealing with thousands of samples to classify. Performing clustering methods allows screening apps to check for signs of hostility in the sample and groups with similar traits. As mentioned earlier, static analysis methods have strengths and weaknesses. Nevertheless, it is unnecessary to use a sandbox to run the sample for a while. Besides, it is not trivial to know the minimum necessary execution time to observe unlawful behavior.

However, our focus is on investigating the features that create a distinction in the grouping of malware and goodware, as well as the impacts of the classification of variants in families. Discovering the value of these attributes, the model can be used for datasets of different sizes. Unlike most articles, we aim to explain the divisions resulting from clustering, to interpret why some versions of the very application can be allocated in different groups. When analyzing malicious

samples from the same family, this verification is even more important, due to the large number of modifications made from common code.

3 Experiments and Results

For the experiments, 111 Scareware applications available on the CICAndMal2017 database of the UNB - University of New Brunswick [15] were analyzed. In addition to the malware samples, several samples of 10 benign applications (*goodware*) were analyzed for feature extraction and pattern determination. The graph of the Fig. 1 displays the names of applications and the distribution of malicious and non-malicious samples.

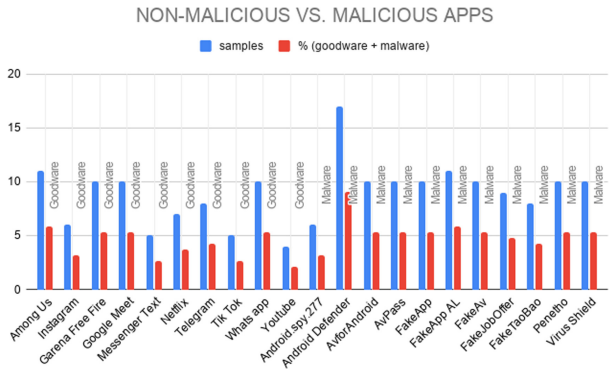

Fig. 1. Information about the non-malicious and malicious dataset. The blue column shows the number of samples of each application/family. The red column shows the percentage to the total dataset (malicious and non-malicious apps). (Color figure online)

We used Androguard [8] to preprocess/unpack the applications. The extracted *xml* files and features values were stored into a database for posterior analysis of similarity and clustering.

We applied the Levenshtein distance [16] to calculate the similarity of malware samples using strings extracted from attributes, set of permissions, activities, services, external classes, and receivers. Levenshtein distance employs the minimum number of insertions, deletions, and substitutions required to transform one string into another.

The Average Linkage algorithm was applied for clustering, a method in which the criterion is the distance of all individuals from one group to each group. For the implementation, we used the `sklearn` library and Python language, in combination with the Maria DB database.

This section presents charts and dendrograms of the groups resulting from sample clustering. Regarding the scale used on the vertical axis (distance), two identical samples have a value of 0. Consequently, the more divergent the two samples are, the greater this value is.

3.1 Analysis of Non-malicious Applications

Non-malicious individuals were grouped using the K-means algorithm (Fig. 3), by analyzing classes, permissions, and activities. It was observed that the applications Tik Tok and Among Us stand out for the greater and lesser amounts of these attributes, respectively. As not all applications had the use of services and receivers, these attributes were not employed in this first experiment.

The tree resulting from the goodware samples clustering is shown in Fig. 2. As can be observed in the dendrogram, the applications were properly grouped. However, copies of Google Meet have been divided into two separate groups (clusters G and H). The Group H samples have fewer attributes of all types when compared to more recent samples, placed in the other group.

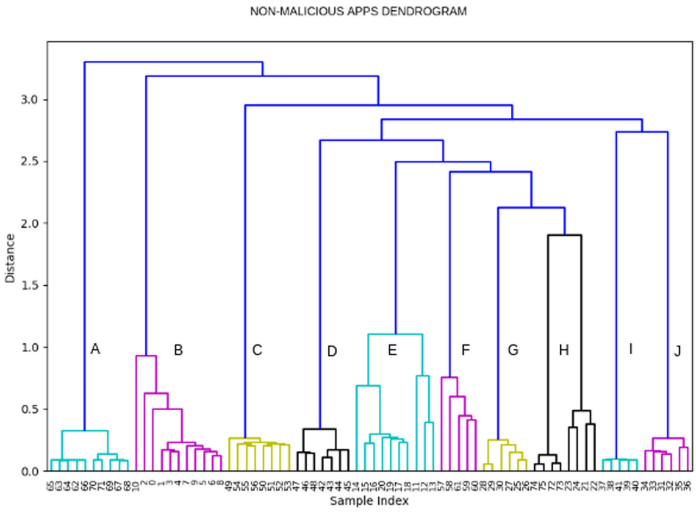

Fig. 2. Dendrogram of all goodware samples.

According to the analysis of distinct and common features, it was noticed that applications in the same cluster had dozens of identical features, and in separate groups, the applications had hundreds of different features. For example, Instagram and Whats app samples had 24 permissions, three services, nine classes, and two activities in common. Comparing the sample feature sets from Instagram and Tik Tok, we found differences in 355 activities, 1238 classes, 66 services, 32 receivers, and 53 permissions.

1. **Whats App - Cluster A**. As differentiators from the other communication applications analyzed, Whats App samples have permissions to receive maps, backup service, NFC, install packages, read phone numbers, and read and write settings. Comparing the 62 and 66 elements, the founded difference was 24 activities, 275 classes and three services, and one permission. The added permission in the last five samples was the `android.permission.READ_PHONE_NUMBERS`. Just by studying the total characteristics of each version, it would not be probable to regard this dissimilarity. There were several variations in size in the samples.
2. **Among Us - Cluster B**. Unlike Garena Free Fire, this game does noDue to space limitations, the list of analyzed samples, the number of features and additional trees weret have multiple attributes with questionable security. Among Us version v2020.6.9 is an outlier of the set. There were some alterations in the samples size.
3. **Telegram - Cluster C.** Samples 37 and 38 were the most similar in the Telegram set. When comparing samples 32 and 39, differences were found in 10 classes, one service, and one receiver. The set of permissions and activities was 100% inherited from the oldest version. About the size, there was an addition in the most recent elements.
4. **Netflix - Cluster D.** The difference in the totals of attributes seems succinct, however, the activities in common with the older version of the set, varied by about 50.5%, while the external classes reached 59% of the variation. The size had an addition in the most recent elements.
5. **Garena Free Fire - Cluster E.** The number of external classes is large. One security question is what is the interest of this game in getting device information about the currently or recently running tasks? Also, this game has other permissions that are considered dangerous, as recording audio, obtaining network status, and Bluetooth. Examining the total of attributes, it is evident a split of the groups because of the dissimilarities of the total numbers of permissions, receivers, and services. The detected alterations between the samples 51 and 60 were three permissions, eight activities, three receivers, seven services, and 62 classes.
6. **Tik Tok - Cluster F** Some differentiated permissions are external storage media reading and access Wifi state. Other several packages are used to read and modify the device's settings. The differences in the Tik Tok samples are perceptible in the feature quantities: an increment in classes, receivers, and activities. These alterations divided the five versions into two groups. When comparing samples 32 and 39, we detected differences of 28 activities, one receiver, 115 classes, and five services. The size had an addition in the most recent elements.
7. **Google Meet - Cluster G and Cluster H.** The divergences between the classes in common with the oldest version varied between 25 and 117 distinct classes. Comparing the elements 22 and 31, for example, we found differences in 12 services, 117 classes, 19 receivers, 38 activities, and 17 permissions. Also, there were some fluctuations in the sample sizes.

8. **Youtube - Cluster H.** The YouTube examples differ due to the growth in classes and decrease activities in the most recent samples. The elements of numbers 6 and 7 had differences of 217 classes, being the least similar pair in the group. The set of permission, service, and receivers was 100% inherited from the version YouTube.v16.01.34. For activity and class sets, despite changes in the total, the inheritance was higher than 83%. Regarding the size, there was a reduction in the most recent elements.
9. **Messenger - Cluster I.** This is the communication application with the highest number of attributes. Messenger elements have several Facebook permissions as write and receive SMS, and call phones. The five versions of Messenger were separated by the number of classes. The set of permissions and receivers was 100% inherited from the oldest version. However, when comparing samples 71 and 75, the discrepancies found were three activities, 607 classes, and one service. Regarding the size, there was an increase in the most recent elements.
10. **Instagram - Cluster J.** The separation of the six Instagram versions can be done based on the addition in the number of services and receivers and the decrease in the number of classes in samples 4 and 5. The added permission in Instagram.v174.0 was the `android.permission.READ_PHONE_NUMBERS` while the activity was the `com.instagram.rtc.activity.RtcGridSandboxActivity`. There were several variations in the number of external classes called in the samples. The three services added in the last two versions were related to background services and the alarm system. The size had an addition in the most recent elements, except in the last one.

Clustering Validation. For the 10 generated clusters, the Precision value was equal to 1. The Recall and F1-Score values were equal to 1 in almost all clusters, except in cluster H where Recall was 0.6 and F1-Score was 0.75.

3.2 Analysis of Malicious Applications

The result of clustering families using K-means is shown in Fig. 3. Due to space limitations, the list of analyzed samples, the number of features and additional trees are in the online Appendix[1]. It was noticed that there was confusion in the samples, both in K-means results and in the groups resulting from hierarchical clustering. In addition, information about unique attributes of each family is in Table 1.

(A) **Android spy 277**. The most appealing permissions are camera, audio recording, SMS messages, and authorization to disable the keylock and any associated password security. In the malware families clustering, the samples from this family were separated into purple and light blue clusters. Analyzing the total of features, samples 3 and 4 would be more similar to each other if compared to the pair 4–5. Meanwhile, samples 3 and 4 have only one activity and ten permissions in common, while the pair 4–5 shares

[1] Online Appendix is available at https://github.com/tsrpimenta/onlineappendix.

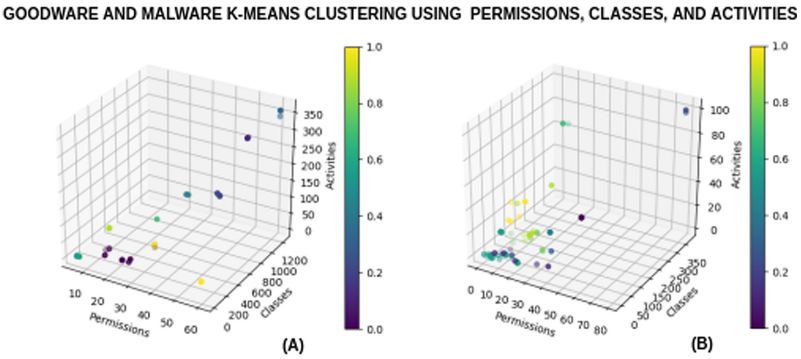

Fig. 3. Results of Goodware(A) and Malware (B) K-Means Clustering using Activities, Classes, and Permissions.

Table 1. Exclusive features of each family.

Family	Permissions	Activities	Receivers	Classes	Services
Android.spy.277	1	14	4	47	2
Android Defender	14	22	9	165	13
AvForAndroid	0	43	9	51	4
AvPass	0	13	2	74	2
FakeApp	0	1	0	1	0
FakeApp AL	2	1	1	26	1
FakeAV	4	5	4	6	7
FakeJobOffer	1	24	4	8	3
FaketaoBao	0	1	4	0	2
Penetho	0	3	0	2	0
Virus shield	3	7	12	59	15

one activity, seven classes, and ten permissions. We emphasize that there were several changes in the size of the samples.

(B) **Android Defender.** This family contains the most extensive number of samples and exclusive features (Table 1). After clustering just the samples from this family, we realized that three groups were formed. The most similar element pairs were 15–17, 8–20, and 9–10. In the clustering of all malware samples, 4 samples from this family were separated into a cluster, demonstrating that they are distinct from the others in this set. We found that this type of malware uses several services and classes with different names, especially for reading and modifying system configurations. This family contains samples with heterogeneous amounts of attributes and sizes.

(C) **AvForAndroid.** This family highlighted out with the highest number of exclusive activities (Table 1). In the clustering, 8 elements were in one cluster and two in the last group. It is interesting to compare samples from

the same family to verify dissimilar behaviors. For example, sample 23 has permissions to change network state, record audio and another 19 different permissions. When, the sample 25 can install shortcuts, get user accounts and get the bookmarks list.

(D) **AVPass.** This family calls several exclusive external classes (Table 1). This family has numerous activities related to books and some related to sniffer actions. Examples are espionage on shopping activities, such as shopping cart, zoom in on products and click on promotions. All samples were in the same group. For the most similar samples of indexes 38, 36, and 39, we detected that what separated element 38 from pair 36–39 were changes in classes and activities.

(E) **FakeApp.** This group has the most simple set of unique attributes. Some of the permissions that stand out are `android.permission.KILL_BACK GROUND_PROCESSES`, `com.android.browser.permission.WRITE_HISTORY_ BOOKMARKS`, and `android.permission.BLUETOOTH_ADMIN`. Analyzing the elements of this family, it can be seen that sample 47 was an outlier, especially due to the increase in the number of permissions. There were several changes in the size of the samples.

(F) **FakeApp AL.** This family has the amplest set of unique classes (Table 1). The samples were divided into two clusters. In sample clustering, the most similar pairs were 54–59 and 60–61. Unusual founded permissions are battery status, access Bluetooth sharing, turn on the device's flashlight, camera, and execute calls.

(G) **FakeAV.** Element 70 was an outlier in this set because of the number of classes. Nevertheless, all samples remained in the same group. Concerning permissions found in this malware were `android.permission.CLEAR_APP_CACHE`, `android.permission.INJECT_ EVENTS`, `android.permission.DELETE_PACKAGES`, `android.permission. INSTALL_PACKAGES`.

(H) **FakeJobOffer.** These samples were grouped with elements of the Android Defender set. Simply by the number of attributes, we could have considered all the samples are equivalent. Nevertheless, the most related pairs were 86–88 and 90–91. The exclusive permission found in this family was the `com.saavn.android.permission.C2D_MESSAGE`.

(I) **Faketaobao.** This family presents few exclusives receivers, services, and activities. Some that stand out are related to boot broadcast, receiving SMS, and activities performing in the background. Although the samples are all in the same cluster, the most divergent elements are copies 83 and 87 and the most similar are pairs 84–89 and 88–90.

(J) **Penetho.** Despite being in the same cluster, the element 92 was separated by the difference in the number of permissions, activities and classes to the other samples. It is interesting to note that these samples do not use receivers and services.

(K) **Virus Shield.** This family has the most significant set of unique receivers (Table 1). Element 107 was an outlier in this set because of the number of classes, therefore being grouped in the same cluster. The pair 103–108 was the least dissimilar in the set. The number of features and the size of each sample were heterogeneous.

Clustering Validation. As shown in Table 2, the values of the validation of the clustering were varied, with the lowest precision values being in the clusters 6, 9, and 11.

Table 2. Calculated metrics (%) for malware clustering results (each cluster is denoted by C followed by its number).

Metric/Cluster	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11
Precision	100	90	89	64	70	45	80	0	37	100	0
Recall	100	100	100	100	100	100	57	0	75	25	0
F1-Score	100	94	94	78	82	62	66	0	50	40	0

3.3 Goodware and Malware Samples Clustering

The K-means clustering of all samples(malware + goodware dataset) is shown in Fig. 4. Due to the size of the dendrogram, the figure can be found in the Online Appendix.

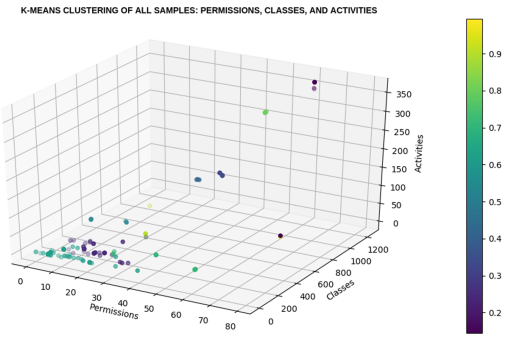

Fig. 4. K-means clustering of all samples.

Most benign apps were well separated, except for a few samples from Google Meet. About malicious samples, the classification of Penetho samples stands out with all samples in the same cluster. We observed that when comparing the familiar attributes between the Garena Free Fire and FakeApp_AL samples, we found thirteen unsafe permissions in common. These permissions have already been mentioned previously in the analyzes of these groups. Unfortunately, the chosen sets of attributes and the similarity measure were not satisfactory for separating families, yet they were capable of distinguishing between malicious and non-malicious samples.

3.4 Clustering Validation

Due to the limitation in the number of pages, the table for validating the clusters of all samples can be found in the Online Appendix. We observed that besides the clusters with values 0, the lowest values of precision, accuracy, and F1-score were those of groups 20, 22. The indices adopted to examine the results of the clusters are shown in the Table 3. According to the outcomes, it was clear that the clustering of non-malicious samples had the best indicators, reaching 0.95 of accuracy. Regarding the grouping of malicious samples families and the total dataset (malware and goodware), some measures revealed that the latter performed better. According to the results of the validation of clustering of malicious samples, we verified that the recall was not satisfying, indicating that although there are exclusive characteristics, the set of equal attributes is also large. This causes the samples to barely distinguishable, creating mixing between groups. Although the non-malicious samples are more distinguishable, the classification of the malicious ones ended up reducing the accuracy of the clustering of all dataset samples.

Table 3. Calculated indices (%) for the goodware, malware and goodware-malware clustering results

Clustering	Accuracy	Precision	Recall	F1-score	Homogeneity	V-Measure
Goodware	95	1	94	96	100	98
Malware	64	83	64	63	72	66
Goodware-Malware	67	80	67	71	89	85

4 Discussion

Concerning Goodware, some questions that remained about similar applications are in different groups. For example, *why did the communication apps stay in separate groups?* The samples of these applications have heterogeneous number of attributes, and some permissions are found in a few applications only (e.g., the Google Meet app has a number of conference-related receivers, such as meeting entry and exit, and data sharing). In the Whats app attribute analysis, we found several backup features, contact blocklists management, two-step authentication functionality, and payment actions. Another finding was that not all applications are allowed to use SMS messages, as noted in Google Meet and Telegram.

Malware vs Goodware. Examining the characteristics of the versions of the non-malicious applications tested, we verified that can be less confused to determine the lineage of the samples, due to a large number of features common among the samples.

Regarding ground truth and labeled datasets, building bases for from benign applications is less complex than for malicious code. The use of antivirus to check whether a file is malicious or not is different from the sample labeling process in

families. In addition, in the analysis of malware, there is still the verification of whether a variable is new or modified code and repackaged.

Static Features. The classification of malware only through static features has limitations, as already mentioned in [21]. Still, permissions are useful tools for detecting dangerous and unwanted behavior (e.g., all analyzed malicious mobile applications had the `KILL_BACKGROUND_PROCESSES`, `REAL_GET_TASKS`, `BLUETOOTH_ADMIN`, and `READ_LOGS` permissions).

During the analysis of the attributes, it was evident that malicious and non-malicious applications use consents potentially harmful to the user's security. However, how do you know what the application intends when it requests permission for recording audio? Would this consent enables the sharing of audio messages among unauthorized people or spying the victim's conversations and use the audios for extortion? Other permissions considered critical appeared in diverse goodware:

- `WRITE_EXTERNAL_STORAGE`: Youtube, Whats app, Telegram, Messenger, Instagram, Garena Free Fire, Netflix, and Tik Tok.
- `ACCESS_FINE_LOCATION`: Youtube, Whats app, Telegram, Messenger, and Instagram.
- `ACCESS_COARSE_LOCATION`: Youtube, Whats app, Telegram, and Messenger.
- `GET_TASKS`: Whats app, Tik Tok, and Garena Free Fire.
- `CHANGE_WIFI_STATE`: Whats app, Messenger, and Garena Free Fire.
- `READ_PHONE_STATE`: Youtube, Whats app, Telegram, Messenger, Instagram, Garena Free Fire, and Netflix.

Some permissions appeared only in malicious code, such as `DISABLE_KEYGUARD` (FaketaoBao, Android.spy.277, FakeApp_AL, Android Defender, and AvForAndroid), and `READ_HISTORY_BOOKMARKS` (FakeApp and AndroidDefender).

Concerning other attributes, we found that families Android Defender and FakeApp_AL have battery monitoring services. The fakeAV and faketaoBao samples presented status and Wifi changes receivers. FakeJobOffer samples have attributes related to audio while AvForAndroid monitors readers of books and pdfs activities. Also, android.spy.277 and FakeApp_AL families are interested in the user's actions with images and videos. Virus Shield and AvPass elements presented payment activities and services.

As a limitation, the similarity measure and the features chosen were insufficient to distinguish malware families and some cases of goodware-malware too.

5 Conclusion

From the experiments presented, it was possible to conclude that malware variants from known families were grouped into separate groups. These results allow us to assume that the feature extraction model is still flawed and demands improvement. Based on these assumptions, the following future works are considered: use of hybrid analysis (static and dynamic features) to describe the most

representative behaviors of malware; automation of the proposed model for large sets of samples acquired incrementally from real environments; and the use of unique attributes in clustering with weights.

References

1. A. F. A. Kadir, N.S., Ghorbani, A.A.: Understanding android financial malware attacks: taxonomy, characterization, and challenges. J. Cyber Secur. Mob. **7**, 1–52 (2018)
2. Alam, S., Traore, I., Sogukpinar, I.: Annotated control flow graph for metamorphic malware detection. Comput. J. **58**, 2608–2621 (2015)
3. Apvrille, A., Strazzere, T.: Reducing the window of opportunity for android malware Gotta catch'em all. J. Comput. Virol. **8**(1–2), 61–71 (2012)
4. Awad, R.A., Sayre, K.D.: Automatic clustering of malware variants. In: IEEE Conference on Intelligence and Security Informatics (ISI) (2016)
5. Burke, D.: Fake covid notification apps and websites aim to steal money and personal data. https://www.cbc.ca/news/canada/nova-scotia/covid-apps-phones-scammers-fraudulent-personal-data-1.5877496
6. Cha, S.H.: Comprehensive survey on distance/similarity measures between probability density functions. City **1**(2), 1 (2007)
7. Connolly, L.Y., Wall, D.S.: The rise of crypto-ransomware in a changing cybercrime landscape: taxonomising countermeasures. Comput. Secur. **87**, 101568 (2019)
8. Desnos, A., et al.: Androguard-reverse engineering, malware and goodware analysis of android applications. google.com/p/androguard 153 (2013)
9. Fedler, R., Schutte, J., Kulicke, M.: On the effectiveness of malware protection on android. Fraunhofer AISEC **45**, 53 (2013)
10. Feizollah, A., Anuar, N.B.R., Salleh, G.S.T., Furnell, S.: AndroDialysis: analysis of android intent effectiveness in malware detection. Comput. Secur. **65**, 121–134 (2017)
11. Felt, A.P., Chin, E., Hanna, S., Song, D., Wagner, D.: Android permissions demystified. In: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, pp. 627–638. ACM (2011)
12. Firdaus, A., Anuar, N., Karim, A., Razak, M.F.A.: Discovering optimal features using static analysis and a genetic search based method for android malware detection. Front. Inf. Technol. Electron. Eng. **19**, 712–736 (2018)
13. Hsiao, S.W., Sun, Y.S., Chen, M.C.: Behavior grouping of android malware family. In: 2016 IEEE International Conference on Communications (ICC), pp. 1–6. IEEE (2016)
14. Karim, M.E., Walenstein, A., Lakhotia, A., Parida, L.: Malware phylogeny generation using permutations of code. J. Comput. Virol. **1**, 13–23 (2005)
15. Lashkari, A.H., Kadir, A.F.A., Taheri, L., Ghorbani, A.: Toward developing a systematic approach to generate benchmark android malware datasets and classification. In: 2018 International Carnahan Conference on Security Technology (ICCST), pp. 1–7 (2018)
16. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. In: Soviet Physics Doklady, pp. 707–710. Soviet Union (1966)
17. Li, Y., Jang, J., Hu, X., Ou, X.: Android malware clustering through malicious payload mining. In: Dacier, M., Bailey, M., Polychronakis, M., Antonakakis, M. (eds.) RAID 2017. LNCS, vol. 10453, pp. 192–214. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66332-6_9

18. 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, pp. 328–332. ACM (2010)
19. Martín, A., Fuentes-Hurtado, F., Naranjo, V., Camacho, D.: Evolving deep neural networks architectures for android malware classification. In: IEEE Congress on Evolutionary Computation (CEC) (2017)
20. Metz, J.: Análise e extração de características estruturais e comportamentais para perfis de malware. Master's thesis, Mestra em Ciências de Computação e Matemática Computacional - USP., São Carlos - SP (Junho 2006)
21. Moser, A., Kruegel, C., Kirda, E.: Limits of static analysis for malware detection. In: Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007), pp. 421–430. IEEE (2007)
22. Nadeem, A.: Clustering malware's network behavior using simple sequential features. Master's thesis, University of Technology, Faculty of Electrical Engineering, Mathematics and Computer Science, September 2018
23. News, S.W.: The covid-19 hoax scareware. https://securitynews.sonicwall.com/xmlpost/the-covid-19-hoax-scareware/. Accessed 25 Mar 2021
24. Perdisci, R., W.Lee, Feamster, N.: Behavioral clustering of http-based malware and signature generation using malicious network traces. In: NSDI, vol. 10, p. 14 (2010)
25. Pfeffer, A., et al.: Malware analysis and attribution using genetic information. In: 7th International Conference on Malicious and Unwanted Software (2012)
26. Poeplau, S., Fratantonio, Y.A., Bianchi, C.K., Vigna, G.: Execute this! Analyzing unsafe and malicious dynamic code loading in android applications. In: NDSS Symposium 2014, pp. 23–26 (2014)
27. Rathore, H., Sahay, S.K., Chaturvedi, P., Sewak, M.: Android malicious application classification using clustering. In: Abraham, A., Cherukuri, A.K., Melin, P., Gandhi, N. (eds.) ISDA 2018 2018. AISC, vol. 941, pp. 659–667. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-16660-1_64
28. Rendón, E., Abundez, I., Arizmendi, A., Quiroz, E.M.: Internal versus external cluster validation indexes. Int. J. Comput. Commun. **5**(1), 27–34 (2011)
29. Samani, R.: Mcafee mobile threat report: Mobile malware is playing hide and steal. https://www.mcafee.com/enterprise/pt-br/assets/reports/rp-quarterly-threats-nov-2020.pdf. Accessed 25 Mar 2021
30. Schmidt, A.D., et al.: Static analysis of executables for collaborative malware detection on android. In: 2009 IEEE International Conference on Communications (2009)
31. Skovoroda, A., Gamayunov, D.: Review of the mobile malware detection approaches. In: 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (2015)
32. Team, N.: What is scareware and how can i avoid it? https://uk.norton.com/norton-blog/2015/09/what_is_scarewarean.html
33. Wüchner, T., Ochoa, M., Pretschner, A.: Malware detection with quantitative data flow graphs. In: Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security (2014)
34. Zhou, Y., Jiang, X.: Dissecting android malware: characterization and evolution. In 2012 IEEE symposium on security and privacy (2012)