Mining Permission Patterns for Contrasting Clean and Malicious Android Applications

Veelasha Moonsamy, Jia Rong, Shaowu Liu

School of Information Technology, Deakin University, 221 Burwood Highway, Vic 3125, Australia.

Abstract

Android application uses permission system to regulate the access to system resources and users' privacy-relevant information. Existing work have demonstrated several techniques to study the *required* permissions declared by the developers, but few attention has been paid for *used* permissions. Besides, no specific permission combination is identified to be effective for malware detection. To fill these gaps, we have proposed a novel pattern mining algorithm to identify a set of contrast permission patterns that aim to detect the difference between clean and malicious applications. In addition, we used a benchmark malware dataset and collected a set of 1227 clean applications to evaluate the performance of the proposed algorithm. Valuable findings are obtained by analyzing the returned contrast permission patterns.

Keywords: Android Permission, Data Mining, Biclustering, Contrast Mining, Permission Pattern

1. Introduction

Between 2010 and 2013, the global telephony industry has witnessed an upsurge in the sales of smartphones. *Smartphone* is a term used to describe a mobile device equipped with enhanced computing capability and connectivity [1], such as *Nexus* by *Google* [2], *iPhone* by *Apple* [3], *Blackberry* by *RIM* [4] and *Windows Phone* by *Microsoft* [5]. A smartphone is usually sold with an in-built Operating System (OS) together with a number of *applications* pre-installed by the device manufacturer. *Application*, the equivalent

Email addresses: v.moonsamy@research.deakin.edu.au (Veelasha Moonsamy), jiarong@acm.org (Jia Rong), swliu@deakin.edu.au (Shaowu Liu)

of software on the PC platform, enhances the smartphone's functionality and supports interactions with the user to accomplish a set of tasks. *Calendar*, *address book*, *alarm clock*, *media player* and *web browser* are the common applications provided by the device manufacturers. More importantly, there exists an application, commonly known as the "*application store*", on every smartphone that provides the smartphone user access to online *application markets* in order to search and download additional applications on their smartphones.

Every device manufacturer hosts an application market for its own smartphone OS, such as *Apple*'s *App Store* [6], *Google*'s *Google Play* [8], *Blackberry*'s *App World* [7]. This type of application market is often referred to as the *official* application market. However, the competitive nature and regulations imposed by *official* application market have led to a surge in the number of independent application markets, also known as *third-party* application markets. Third-party application markets are till date a popular choice amongst smartphone users because of the availability of a large variety of free/low-priced applications and the inexistence of application vetting process. Nonetheless, one cannot guarantee the cleanliness of such applications [9].

To effectively detect malicious applications (also known as *malware*), many efforts have been contributed, in the last five years, into studying the nature of smartphone platforms and their applications. In addition to being the leading smartphone platform, *Android* is also the most infected OS due to an exponential growth in the number of malware deployed on the application markets. As one of its security measures, the *Android* platform employs a permission system to restrict applications' privileges in order to secure user privacy-relevant resources [12]. An application needs to get the user's approval for the requested permissions to access these restricted resources. Thus, the permission system was designed to protect a user from applications with invasive behaviors, but its effectiveness highly depends on the user's comprehension of permission approval. We refer to the permissions that are requested during application installation as *required* permissions.

Unfortunately, not all the users read or understand the warnings of *required* permissions shown during application installation. To improve this situation, many researchers have tried to interpret *Android* permissions and their combinations [13–16]. Frank et al. [12] proposed a probability model to identify the common *required* permission patterns for all *Android* applications. Zhou and Jiang [11] listed the top *required* permissions for both clean and malicious applications, but only individual permissions were considered by frequency counting. The problem of identifying patterns in permission

combinations that can provide better performance for malware detection still remains.

Furthermore, in the existing literature, only *required* permissions have been considered in permission pattern mining and no work has incorporated *used* permissions, which can be described as follows: Whenever an Application Programming Interface (API) call is invoked during the execution of an application, the smartphone OS will verify if the API call is permission-protected before proceeding to execute the call; such permissions are referred to as *used* permissions. To our knowledge, we are the first to consider both the *required* and *used* permissions that are extracted from static analysis by the *Andrubis* system (http://anubis.iseclab.org) [17]. Accordingly, our aim is *to propose an efficient pattern mining method to identify a set of contrast permission patterns that effectively distinguish malware from the clean applications.*

In order to use pattern mining technique to identify the desired permission patterns, we utilized a clean dataset and a malicious one. In 2012, Zhou and Jiang [11] published the first benchmark dataset of malicious applications in 49 malware families and was collected from third-party markets between August 2010 to October 2011. This is an ideal malware dataset for our experiment as it includes an extensive list of malware families collected over 14 months. On the other hand, due to the unavailability of a clean dataset published during the same time period as Zhou and Jiang's, we collected our own set of clean applications. Those clean applications were downloaded from two popular third-party *Android* applications markets: *SlideME* (http://slideme.org) and *Pandaapp* (http://android.pandaapp.com). We sorted the clean applications based on their upload dates and the ratings given by the users, and only the top ones were selected. Each application was scanned by forty-three antivirus engines on *VirusTotal* [18], and only those that passed all virus tests were considered as "clean" and included in our clean dataset. Similar to Zhou and Jiang, we represent applications in the collected clean dataset using a vector of 130 binary values, each of which is associated with one of the 130 official *Android* permissions. A value 1 is assigned to a permission only if it is required or used by an application, otherwise, 0 is given instead.

The novelty and contributions of this work can be summarized as follows:

- We proposed a *Contrast Permission Pattern Mining* algorithm to identify the interesting permission sets as the patterns can be used to distinguish malicious applications from clean ones.

- Beyond the current studies that focused on *required* permissions only,

we also considered the *used* permissions and our experimental work showed that both *required* and *used* permissions were important to be considered in late malware detection task.

- We collected a new dataset that contains 1227 clean applications that were uploaded to third-party markets from August 2010 to October 2011.
- We utilized a hierarchical *Biclustering* method to initially analyze both clean and malware datasets. The resulting figures provided a straightforward visualization of the data distribution, from which we built up our model of mining a set of permissions rather than using individual permissions as the patterns.

The rest of the paper is organized as follows: Section 2 briefly reviews the concepts of the *Android* platform, its applications, the permission system and the current research work in malware detection. In Section 3, we present our initial analysis on the collected datasets using statistical method and biclustering followed by our proposed contrast pattern mining algorithm. The experiments and the empirical results are then reported in Section 4 followed by a further discussion on our findings. Finally, Section 5 concludes the entire paper together with our future work.

2. Background and Related Work

2.1. Android

Android is a Linux-based OS, which was designed and developed by the *Open Handset Alliance* in 2007 [19]. The *Android* platform is made up of multiple layers consisting of the OS, Java libraries and basic built-in applications [20]. Additional applications can be downloaded and installed from either official or third-party markets.

Google provides the application developer community with a *Software Development Kit* (SDK) [21] to build *Android* applications. The SDK includes a collection of Java libraries and classes, sample applications and developer documentations. The SDK can be used as a plug-in for Eclipse IDE [22] and therefore allows developers to code their applications in a rich Java environment. One particularly useful feature of the SDK is the *Android emulator* which allows developers to test their applications in virtual devices supporting various versions of *Android*.

An *Android* application includes two folders and one file: (i) *Class*, (ii) *Resources* and (iii) *AndroidManifest.xml*. The *Class* folder contains the

application's source code in Java; the *Resources* folder stores the multimedia files; and the *AndroidManifest.xml* file lists the *required* permissions that are declared by the developer. When the Java source code is ready, it is then compiled and converted into Dalvik byte code [23] and bundled with the *Resources* folder and *AndroidManifest.xml* file to generate the *Android Application Package* (APK). Finally, before the APK can be installed on a device or emulator, the developer has to generate a key and sign the application.

Android developers can upload their applications to either the official market, *Google Play* [24], or any third-party market. To secure the privacy-relevant resources for its users, *Google* provides automatic antivirus scanning. The applications will be rejected from *Google Play* if any malicious content is detected. From 2012, *Google* has extended its antivirus service on the new *Android* 4.2 OS, which is claimed to be able to scan applications before they are installed on the device [26].

2.1.1. Android Permission System

Google applies the permission system as a measure to restrict access to privileged system resources. Application developers have to explicitly mention the permissions that need user's approval in the *AndroidManifest.xml* file. *Android* adopts an 'all-or-nothing' permission granting policy. Hence, the application is installed successfully only when the user chooses to grant access to all of the *required* permissions.

There are currently 130 official *Android* permissions and are categorized into four types: *Normal*, *Dangerous*, *Signature* and *SignatureOrSystem* [27]. *Normal* permissions do not require the user's approval but they can be viewed after the application has been installed. *Dangerous* permissions require the user's confirmation before the installation process starts; these permissions have access to restricted resources and can have a negative impact if used incorrectly. A permission in *Signature* category is granted without the user's knowledge only if the application is signed with the device manufacturer's certificate. The *SignatureOrSystem* permissions are granted only to the applications that are in the *Android* system image or are signed with the device manufacturer's certificate. Such permissions are used for special situations where the applications, built by multiple vendors, are stored in one system image and share specific features.

After an application is installed, a set of APIs will be called during the runtime. Each API call is associated with a particular permission. When an API call is made, the *Android* OS checks whether or not its associated permission has been approved by the user. Only a matching result will

lead to the execution of the API call. In this way, the *required* permissions are able to protect the user's private information from unauthorized access. However, an API call invocation cannot fully stop the malware developers from declaring *required* permissions for their applications. Based on the above observation, several studies have tried to identify the common *required* permissions that are frequently declared by *Android* application developers.

By applying the *Self-Organizing Map* (SOM) algorithm, Barrera et al. [13] studied the trends of permission requests from a dataset of $1,100$ applications downloaded from the official market. Frank et al. [12] selected $188,389$ applications from the official market and analyzed the combinations of permissions requests by these applications. A probabilistic method was proposed to deduce the popular permission patterns based on the applications' popularity (ratings together with number of reviews), that is, the deviation of permission requests for high- and low-ranked applications. Bartel et al. [28] proposed an automated tool that can statistically analyze the methods defined in an application and subsequently generate the permissions required by the application. This in turn ensures that the user does not grant access to unnecessary permissions when installing the application. A model designed by Sanz et al. [29] is based on features which comprised solely of *Android* permissions.

The aforementioned existing work studied the applications that were collected mainly from the official market. The results and findings help us to understand the *Android* permission system and the patterns for normal permission requests. However, compared to clean applications, we are more interested in the unusual permission requests, which are considered as more valuable for the detection of *Android* malware.

2.2. Android Malware Detection with Permissions

Malware detection within the *Android* platform is gaining a fair amount of attention from researchers in both academia and industry; however, there is a lack of work on the detection of *Android* malware using permission patterns. Rassameeroj and Tanahashi [30] used visualization techniques and clustering algorithms to reveal normal and abnormal permission request combinations. They evaluated their methodology on a dataset comprising of 999 applications. After analyzing the extracted permission combinations, they claimed that nearly 8% of the applications were potential malicious.

Chia et al. [16] argued that the current user-rating system was not a reliable source of measurement to predict whether or not an application was malicious. Their dataset consisted of 650 applications from the official market and $1,210$ applications from a third-party market. The *required* permissions

were extracted from the dataset, together with other application-related information to develop a risk signal mechanism for detecting malware.

Sahs and Khan [31] focused on feature representation as one of the challenges to malware detection. The features to be represented included: (i) permissions extracted from manifest files and (ii) control flow graphs for each method in an application. Each feature was processed independently using multiple kernels and applied a one-class *Support Vector Machine* to train the classifiers. However, the evaluation results showed that the common features existing in both clean and malware datasets cause detection error rate.

Wu et al. [32] put forward a static feature-based technique that can aid towards malware detection. First, they apply *K-means* algorithms to generate the clusters and use *Singular Value Decomposition* to determine the number of clusters. In the second step, they classify clean and malicious applications using the *k-Nearest Neighbor* (kNN) algorithm.

Zhou et al. [33] proposed a two-layered system known as, *DroidRanger* and makes use of "permission-based behavioral foot-printing and heuristics-based filtering". The authors observed that the permissions extracted from the *AndroidManifest.xml* files of malicious applications gave an insight into uncommon permission requests by some malware families. In Sanz et al.'s work [29], they extracted the permissions and the hardware features to build the feature set. As a result, clean applications require two to three permissions on average, but some of malicious applications only have one permission and are still able to carry out the attack.

Most of the work extracted a feature set to represent the applications. The information carried by those features were different from work to work. There is no evidence to show which features give the best detection result; nonetheless, each study considered only the *required* permissions. Accordingly, we consider taking the permissions as the only features to represent the applications and expect to find specific permission patterns to show the difference between clean and malicious applications.

2.3. Summary

Malware proliferation is rising exponentially and the attack vectors used by malware authors are getting more sophisticated. Current solutions proposed to thwart attacks by malicious applications will struggle to keep up with the increase of malware. The *Android* platform relies heavily on its permission system to control access to restricted system resources and private information stored on the smartphone. As demonstrated by [12], [34],

and [35], permissions that are requested by applications during installation can be helpful in identifying permission patterns.

However, we identify the following problems in the existing literature:

Problem 1 What *required* permission patterns can be used to detect malicious applications?

Problem 2 What *used* permission patterns can be used to detect malicious applications?

Problem 3 Can we extract useful information by incorporating *used* permissions into the permission patterns?

Problem 4 What method can we use to identify these expected permission patterns?

In this paper, we aim to extend the current statistical method used for identifying permission patterns in *Android* applications by applying pattern mining techniques to a set of clean and malicious applications in order to better understand the similarities and differences between these two datasets. With the help of visualization graphs, we establish possible connections between *required* permissions and *used* permissions in order to extrapolate emerging permission patterns that are frequently requested by applications. Then, we apply contrast set mining on the permissions patterns from clean and malicious applications to identify which patterns are most prevalent in each dataset.

3. Mining Permission Patterns

The most common method used to analyze *Android* permissions are statistical-based; for instance, frequency counting by Zhou and Jiang [11] and probabilistic model by Frank et al. [36]. Thus, we started our work by performing an initial analysis on the clean and malware datasets using frequency counting and extended it to incorporate *used* permissions. To further extend the work of Barrera et al. [13] who utilized SOM for application clustering and visualization, we applied the *biclustering* algorithm to not only group the applications but also their respective permissions. Finally, a novel contrast permission pattern mining algorithm is presented to identify specific permission patterns that can help distinguish between clean and malicious applications.

3.1. Classic Statistical Analysis on Android Permissions

We performed a statistical analysis to study both the *required* and *used* permissions for clean and malicious applications. Hence, the following four sub-datasets were extracted: (1) *Required* permissions for clean applications; (2) *Required* permissions for malicious applications; (3) *Used* permissions for clean applications; and (4) *Used* permissions for malicious applications. Direct frequency counting is employed on all four sub-datasets to find out the most popular permissions required or used by clean and malicious applications.

Table 1: Top 20 Required Permissions by Clean and Malicious Applications

Clean Applications		Malicious Applications	
Required Permission	Frequency	Required Permission	Frequency
INTERNET	1121 (91.36%)	INTERNET	1199 (97.72%)
ACCESS_NETWORK_STATE	663 (54.03%)	ACCESS_COARSE_LOCATION	1146 (93.40%)
READ_PHONE_STATE	391 (31.87%)	VIBRATE	994 (81.01%)
WRITE_EXTERNAL_STORAGE	362 (29.50%)	WRITE_EXTERNAL_STORAGE	823 (67.07%)
ACCESS_COARSE_LOCATION	236 (19.23%)	READ_SMS	779 (63.49%)
VIBRATE	210 (17.11%)	WRITE_SMS	762 (62.10%)
WAKE_LOCK	188 (15.32%)	READ_CONTACTS	680 (55.42%)
ACCESS_FINE_LOCATION	162 (13.20%)	BLUETOOTH	633 (51.59%)
GET_TASKS	125 (10.19%)	WRITE_CONTACTS	542 (44.17%)
SET_WALLPAPER	102 (8.31%)	DISABLE_KEYGUARD	491 (40.02%)
ACCESS_WIFI_STATE	64 (5.22%)	WAKE_LOCK	471 (38.39%)
RECEIVE_BOOT_COMPLETED	60 (4.89%)	RECORD_AUDIO	461 (37.57%)
READ_CONTACTS	58 (4.73%)	ACCESS_FINE_LOCATION	446 (36.35%)
WRITE_SETTINGS	45 (3.67%)	ACCESS_NETWORK_STATE	416 (33.90%)
CAMERA	43 (3.50%)	READ_PHONE_STATE	414 (33.74%)
CALL_PHONE	42 (3.42%)	SET_ORIENTATION	413 (33.66%)
SEND_SMS	34 (2.77%)	CHANGE_WIFI_STATE	384 (31.30%)
RESTART_PACKAGES	32 (2.61%)	READ_LOGS	361 (29.42%)
RECEIVE_SMS	31 (2.53%)	BLUETOOTH_ADMIN	342 (27.87%)
RECORD_AUDIO	27 (2.20%)	RECEIVE_BOOT_COMPLETED	325 (26.49%)

After comparing the top 20 *required* permissions for clean and malicious applications listed in Table 1, we found that the malicious applications requested a total of $14,758$ permissions, which was much more than clean applications ($4,470$ permissions). Among these permissions, we found some of them only appeared in one dataset, that is, those permissions are only requested or used by clean applications but not malicious ones, and vice versa. We call these permissions as 'unique permissions'. Similarly, we name those permissions that appear in both clean and malware datasets as 'common permissions'. Totally, there are 33 unique *required* permissions for clean applications and 20 for malicious ones; and also 70 common *required* permissions. Another 5 permissions have never been requested by any application. For *used* permissions, 9 unique ones for clean applications and only 4 for malicious ones. The number of common *used* permissions dropped to 28, and a large number of 87 permissions have never been used by any application. Four common permissions that were most frequently

required by both clean and malicious applications are as follows: `INTERNET`, `ACCESS_COARSE_LOCATION`, `WRITE_EXTERNAL_STORAGE` and `VIBRATE`. In contrast, there were nine out of twenty *required* permissions appeared frequently in malware dataset than in clean one. Moreover, when comparing the top 20 *used* permissions in clean and malicious applications in Table 2, we observed that sixteen out of twenty popular *used* permissions are common in both datasets.

Table 2: Top 20 Used Permissions by Clean and Malicious Applications

Clean Applications		Malicious Applications	
Used Permission	Frequency	Used Permission	Frequency
INTERNET	1029 (83.86%)	INTERNET	1161 (94.62%)
WAKE_LOCK	816 (66.50%)	ACCESS_COARSE_LOCATION	1125 (91.69%)
ACCESS_NETWORK_STATE	738 (60.15%)	VIBRATE	954 (77.75%)
VIBRATE	608 (49.55%)	WAKE_LOCK	826 (67.32%)
READ_PHONE_STATE	457 (37.25%)	ACCESS_WIFI_STATE	584 (47.60%)
ACCESS_COARSE_LOCATION	372 (30.32%)	ACCESS_NETWORK_STATE	519 (42.30%)
SET_WALLPAPER	126 (10.27%)	READ_SMS	473 (38.55%)
ACCESS_FINE_LOCATION	116 (9.45%)	WRITE_CONTACTS	426 (34.72%)
GET_ACCOUNTS	98 (7.99%)	READ_PHONE_STATE	354 (28.85%)
ACCESS_WIFI_STATE	85 (6.93%)	RECORD_AUDIO	319 (26.00%)
READ_SMS	82 (6.68%)	SET_WALLPAPER	297 (24.21%)
RESTART_PACKAGES	65 (5.30%)	ACCESS_FINE_LOCATION	199 (16.22%)
GET_TASKS	61 (4.97%)	GET_ACCOUNTS	178 (14.51%)
CHANGE_CONFIGURATION	55 (4.48%)	GET_TASKS	124 (10.11%)
RECEIVE_SMS	37 (3.02%)	RECEIVE_BOOT_COMPLETED	111 (9.05%)
FLASHLIGHT	37 (3.02%)	ACCESS_CACHE_FILESYSTEM	101 (8.23%)
WRITE_CONTACTS	34 (2.77%)	WRTIE_OWNER_DATA	59 (4.81%)
RECEIVE_BOOT_COMPLETED	23 (1.87%)	CHANGE_CONFIGURATION	52 (4.24%)
WRTIE_OWNER_DATA	12 (0.98%)	READ_HISTORY_BOOKMARKS	49 (3.99%)
WRITE_SETTINGS	10 (0.81%)	EXPAND_STATUS_BAR	41 (3.34%)

Statistical method such as direct frequency counting is suitable for identifying single permissions that are popular in each sub-datasets respectively. However, it still requires further manual checking to confirm the obtained permission lists for clean and malicious applications. This, in turn, further complicates the counting process if permission combinations are to be considered instead of individual permissions. Therefore, we continued our analysis of *Android* permissions by applying the *biclustering* algorithm as a next step in order to provide visualization of the relationship between permissions and applications.

3.2. Visualization Using Biclustering

Biclustering [37] is a special cluster analysis method, which applies classic clustering to both rows and columns, simultaneously, in a two-dimensional data matrix. In this work, *biclustering* can help group applications that request or use different permission combinations as well as show us the clusters of permission combinations based on the various applications associated with them.

The *biclustering* is achieved by performing *Agglomerative Hierarchical Clustering*(AHC) [38, 39] on both dimensions of the data matrix. The *AHC* is first applied along the columns of the data matrix and then applied along the rows of the row-clustered data matrix.

Unlike the common clustering methods which identify a single set of clusters, the *AHC* is a bottom-up clustering method that seeks to identify clusters with sub-clusters. It forms a multilevel hierarchical clustering tree, where lower-level clusters are joined to form higher-level clusters. The steps can be described as follows:

Step 1: Generate the Disjoint Clusters - The *AHC* starts with every single data object, i.e. each data object is assigned to a separate cluster.

Step 2: Form a Distance Matrix - The pairwise distances between the disjoint clusters are calculated using the *Ward's linkage* [40] and are used to initialize the distance matrix.

Step 3: Merge two Closest Clusters - Based on the distance matrix, each cluster is merged with the next closest cluster with the shortest distance.

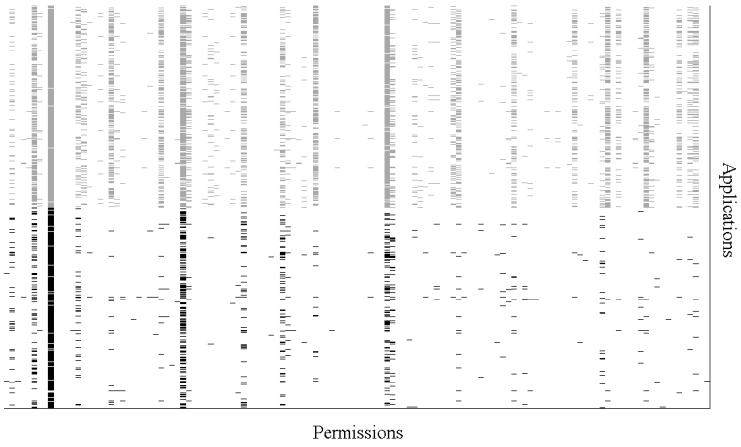
Step 4: Update the Distance Matrix - After the merging, the distance matrix needs to be updated by calculating the new distance between each pair of merged clusters.

Step 5: Obtain the Hierarchical Clustering Tree - Steps 3 and 4 are repeated until all clusters are merged into one large single cluster, resulting in a complete hierarchical clustering tree.

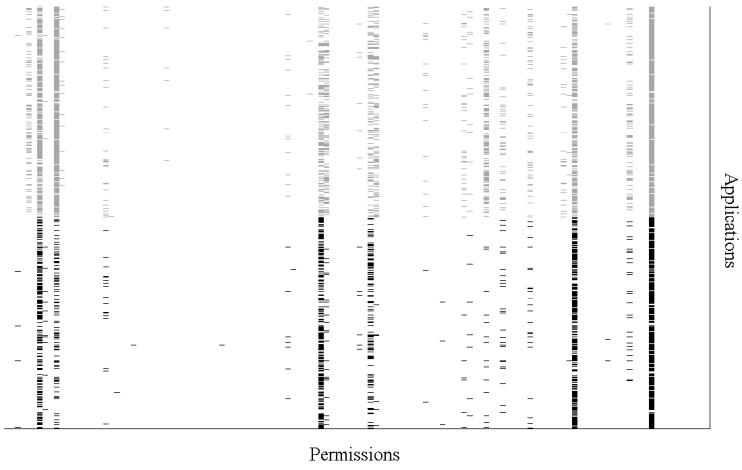
We applied the above *biclustering* steps on two separate sub-datasets extracted from the malware dataset and our collected clean one. One sub-dataset contains all clean and malicious applications with *required* permissions (see Fig. 1a); the second sub-dataset includes the same applications but with *used* permissions (see Fig. 1b).

As a result, we had two output matrices - the *required* permissions are shown in Fig. 2 and the *used* permissions are in Fig. 3. Based on the statistical analysis presented in Section 3.1, there exist unique and common permissions for either clean or malicious applications. As binary values are shown in these matrices, we manually picked up four colors to mark the values to visualize different types of permissions for clean and malicious applications:

- Dark Green shows *Common permissions* for clean applications;



(a) Required Permissions



(b) Used Permissions

Figure 1: Visualization of Sub-datasets for Biclustering: white for 0s; gray for 1s in clean applications; and black for 1s in malicious applications

- Sandy Brown shows *Common permissions* for malicious applications;
- Blue indicates *Unique permissions* for clean applications; and
- Red indicates *Unique permissions* for malicious applications.

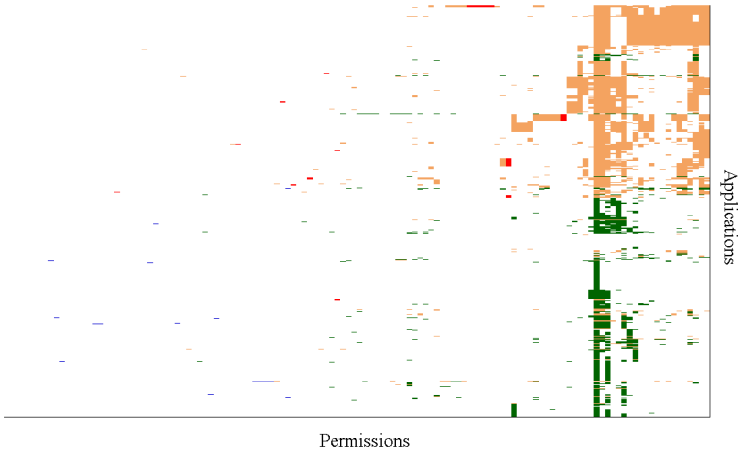


Figure 2: Resulting Matrix for Required Permissions by Biclustering

From these two figures, we can easily observe that more permissions are declared as required than those that are actually used. This further confirms our results from the statistical analysis in Section 3.1. Furthermore, generally, malicious applications either request or use more permissions than clean applications. In normal cases, the unique permissions should perform better than the common permissions to contrast between clean and malicious applications. However, from the resulting matrices, we only have few unique permissions shown in bright red and blue colors compared to the large set of common permissions. Therefore, when applying only statistical methods, it is easy to ignore those unique permissions because of their low frequency. Furthermore, it is obvious to see the color blocks in both figures, which indicate specific permission combinations have potential capability to group applications in separate clusters. The aforementioned observations give rise to the following challenges, which are addressed in the rest of the paper: How can we use the unique permissions for contrast detection? How can we find out the permission combinations and use them as the patterns for malware detection?

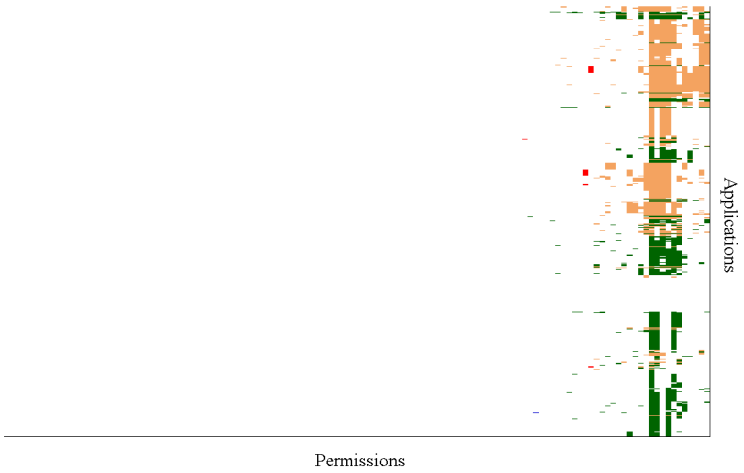


Figure 3: Resulting Matrix for Used Permissions by Biclustering

3.3. Contrast Permission Pattern Mining

In order to tackle the two challenges presented in Section 3.2, the *Contrast Permission Pattern Mining* (CPPM) is proposed. The output permission patterns are expected to have the ability to indicate the difference between the clean and malicious applications. *CPPM* was designed to process more than one dataset and take both common and unique permissions and their combinations into consideration. Two major processes are involved in *CPPM*: (1) candidate permission itemset generation, and (2) contrast permission pattern selection.

3.3.1. Candidate Permission Itemset Generation

The purpose of this process is to obtain a number of candidate permission combinations that are most likely to be the expected contrasting patterns. *CPPM* takes at least two datasets as input. In this case two datasets are loaded, each of which contains either clean or malicious applications. We generate the candidate permission itemsets for every dataset using the same procedure, which can be described as the following two steps:

Apriori-based Itemset Enumeration

Given Dx is one of the input datasets with either *required* or *used* permissions, which contains n *Android* applications. Let $I = \{A, B, C \dots\}$ be the set of possible items (or permissions requested or used by the

applications) in Dx. The *Apriori-based* approach [41], enumerates candidate itemset from the simplest structure with only a single item. Based on this single item, a more complex itemset is then obtained by adding new items. This joining operation is repeated continuously to increase the number of the items in the itemsets. In each iteration, only one item is added into the existing candidate itemset. One item will not appear twice in one itemset. However, the *Apriori-based* approach can generate a large number of candidate itemsets with high computational cost. To solve this problem, we employ a support-based pruning technique to reduce the number of candidate itemsets and consequently, the experimental time.

Support-based Candidate Pruning

Support is usually used to measure the occurrence frequency of a certain item or itemset in a dataset. Let $A,\ B \subseteq I$ be two items, and $\{A, B\}$ forms a candidate itemset. The support of the candidate itemset $\{A, B\}$ can be calculated by:

$$supp(A, B) = \frac{number\ of\ applications\ that\ contain\ A\ and\ B\ in\ Dx}{total\ number\ of\ applications\ in\ Dx} \tag{1}$$

The candidate itemset $\{A, B\}$ is considered as *frequent* only if $supp(A, B) \geq \delta_{supp}$, where δ_{supp} is user-specified minimum *support* threshold. In classic pattern mining methods, only the frequent itemset is considered. Any itemset with a lower support than the pre-determined threshold is treated as *infrequent* and discarded immediately. However, in our case, both the statistical analysis and *biclustering* results show most of the unique permissions are requested or used by few applications. This is an indication that their support values are definitely low. In order to inadvertently miss any valuable patterns, we decide to take both frequent and infrequent candidate itemsets, but only use frequent ones to generate new candidate itemsets to cut down the computational cost.

3.3.2. Contrast Permission Pattern Selection

The permission itemsets generated from Section 3.3.1 need to be reduced according to the pre-defined selection criteria. This process guarantees that the output itemsets are highly contrasted between clean and malicious applications. The contrasting permission patterns reflect the different behaviors present amongst the applications in the two datasets. If one permission

itemset is frequent in one dataset, it is often considered to carry more common features than the infrequent ones. Therefore, the selection of specific contrast permission pattern is based on comparison of its support in both datasets. The bigger the difference in support values, the greater the contrast a permission pattern has.

Given one candidate permission itemset $\{A, B\}$ and its support values in clean and malware datasets, $supp(A, B)_{clean}$ and $supp(A, B)_{malicious}$, calculate the difference by $diff(A, B) = supp(A, B)_{clean} - supp(A, B)_{malicious}$. $\{A, B\}$ is identified as a contrasted permission pattern only if $diff(A, B) \geq \delta_{diff}$, where δ_{diff} is a user-specified *minimum support difference*. All the candidate permission itemsets need to be tested using this approach, and the ones with big support difference will be selected as the final output contrast permission patterns.

4. Experiments and Results

4.1. Experiment Settings

According to the statistical analysis and *biclustering* resulting figures, not all the permissions are required or used. In the experiment to evaluate the proposed Contrast Permission Pattern Mining algorithm, we ignore the permissions that are not required or used in each sub-datasets respectively. Table 3 gives more details of the four new sub-datasets.

The statistical analysis results also show that only a small set of permissions have support that are greater than 0.1 (10%), so we follow the previous studies [42–44] to set 0.05 as an acceptable value for minimum support threshold for all four sub-datasets in *CPPM*. The minimum support difference threshold is set to be 0.15 (15%) and applied to filter out itemsets that are highly contrasted between clean and malicious applications.

Table 3: Four Sub-datasets Used in Contrast Permission Pattern Mining Experiments

	Dataset	Permission involved	Permission Discarded
1	Clean_Required	103	27
2	Malicious_Required	90	40
3	Clean_Used	37	93
4	Malicious_Used	31	99

Table 4: Permission Index

Permission Category	Permission ID	Permission Name
Normal	*pms*0001	INTERNET
Normal	*pms*0006	ACCESS_NETWORK_STATE
Normal	*pms*0007	VIBRATE
Normal	*pms*0012	RESTART_PACKAGES
Normal	*pms*0013	RECEIVE_BOOT_COMPLETED
Normal	*pms*0023	ACCESS_WIFI_STATE
Dangerous	*pms*0002	ACCESS_FINE_LOCATION
Dangerous	*pms*0003	WAKE_LOCK
Dangerous	*pms*0004	WRITE_EXTERNAL_STORAGE
Dangerous	*pms*0005	READ_PHONE_STATE
Dangerous	*pms*0008	READ_CONTACTS
Dangerous	*pms*0011	READ_LOGS
Dangerous	*pms*0020	ACCESS_COARSE_LOCATION
Dangerous	*pms*0021	SEND_SMS
Dangerous	*pms*0022	GET_TASKS
Dangerous	*pms*0024	CHANGE_WIFI_STATE
Dangerous	*pms*0028	WRITE_CONTACTS
Dangerous	*pms*0029	RECEIVE_SMS
Dangerous	*pms*0030	READ_SMS
Dangerous	*pms*0031	WRITE_SMS
Dangerous	*pms*0036	CALL_PHONE
Signature	*pms*0010	FACTORY_TEST
SignatureOrSystem	*pms*0052	INSTALL_PACKAGES

4.2. Contrast Permission Patterns

Among the permission patterns that were generated, we found that 23 distinct permissions were present in the highly contrasted permission combinations as listed in Table 4. We classified the permissions based on the following permission categories: *Normal*, *Dangerous*, *Signature* and *SignatureOrSystem*. We recorded 6 permissions belonging to the *Normal* category, 15 permissions for the *Dangerous* category and 1 permission each for the *Signature* and *SignatureOrSystem* category.

We found that the generated permission combinations are correlated and differed between clean and malicious applications. Based on the experimental results, we recorded 56 *required* permission patterns that are unique to the malware dataset, 31 *used* permission patterns that only appear amongst malware, 17 *required* permission patterns and 9 *used* permission patterns that are present in both clean and malware dataset. These findings are formed as permission combinations which are listed in Table 5 - 13, and summarized with respect to the usage type of the permission in the following section. First, we define the four types of permission combinations as follows:

(i) **Unique Required Permission (URP)**: *Required* permission patterns present only in malware dataset

(ii) **Unique Used Permission (UUP)**: *Used* permission patterns present only in malware dataset

(iii) **Common Required Permission (CRP)**: *Required* permission patterns present in both clean and malware datasets

(iv) **Common Used Permission (CUP)**: *Used* permission patterns present in both clean and malware datasets

4.2.1. Unique Required Permission (URP) Patterns

From Table 5 - 10, we present the permission patterns that were frequently required by the applications in our dataset. It is worth noted that these *required* permission patterns were unique to the malware dataset only; hence the support value for the clean applications is 0.

Table 5: Unique Required Permission Sets in Malware Dataset (*Normal Permissions*)

Permission Set	Support		Permission Set ID
	Clean	*Malware*	
$pms0001$, $pms0005$, $pms0023$	0	0.6309	$URPSet_1$
$pms0001$, $pms0006$, $pms0023$	0	0.6031	$URPSet_2$
$pms0001$, $pms0013$	0	0.5542	$URPSet_3$
$pms0006$, $pms0013$	0	0.5168	$URPSet_4$
$pms0006$, $pms0031$	0	0.4964	$URPSet_5$
$pms0001$, $pms0021$	0	0.4312	$URPSet_6$
$pms0013$, $pms0023$	0	0.4263	$URPSet_7$
$pms0021$, $pms0029$	0	0.3701	$URPSet_8$
$pms0004$, $pms0013$	0	0.3660	$URPSet_9$
$pms0001$, $pms0005$, $pms0020$	0	0.3562	$URPSet_{10}$
$pms0001$, $pms0005$, $pms0006$, $pms0007$	0	0.3497	$URPSet_{11}$
$pms0001$, $pms0004$, $pms0020$	0	0.3122	$URPSet_{12}$
$pms0023$, $pms0024$	0	0.3097	$URPSet_{13}$
$pms0006$, $pms0008$	0	0.2975	$URPSet_{14}$
$pms0013$, $pms0031$	0	0.2943	$URPSet_{15}$
$pms0006$, $pms0036$	0	0.2869	$URPSet_{16}$
$pms0013$, $pms0021$	0	0.2804	$URPSet_{17}$
$pms0007$, $pms0036$	0	0.2494	$URPSet_{18}$
$pms0012$, $pms0021$	0	0.2405	$URPSet_{19}$
$pms0013$, $pms0036$	0	0.2380	$URPSet_{20}$
$pms0006$, $pms0012$	0	0.2372	$URPSet_{21}$
$pms0012$, $pms0029$	0	0.2282	$URPSet_{22}$
$pms0012$, $pms0013$	0	0.2234	$URPSet_{23}$
$pms0012$, $pms0036$	0	0.2119	$URPSet_{24}$
$pms0001$, $pms0004$, $pms0005$, $pms0007$	0	0.2014	$URPSet_{25}$

In Table 5, we list the top 25 permission combinations where the first permission in the listed patterns belongs to the *normal* permissions category. The permission combinations from $URPSet_1$ and $URPSet_2$ were both required by more than 60% of the malware. In fact, we found that the `INTERNET` permission ($pms0001$) is frequently requested along with other permissions and their support value are relatively high. The permission combination, `INTERNET` and `RECEIVE_BOOT_COMPLETED` were present in 55% of the malware dataset. Other such patterns involving the `INTERNET` permission are listed in Table 5.

Table 6: Unique Required Permission Sets in Malware Dataset (*ACCESS_FINE(COARSE)_LOCATION*)

Permission Set	Support		Permission Set ID
	Clean	*Malware*	
$pms0002$, $pms0005$, $pms0020$	0	0.2690	$URPSet_{26}$
$pms0002$, $pms0004$, $pms0020$	0	0.2576	$URPSet_{27}$
$pms0002$, $pms0005$, $pms0023$	0	0.2307	$URPSet_{28}$
$pms0002$, $pms0004$, $pms0023$	0	0.2234	$URPSet_{29}$

Table 7: Unique Required Permission Sets in Malware Dataset (*SMS*)

Permission Set	Support		Permission Set ID
	Clean	*Malware*	
$pms0030$, $pms0036$	0	0.3228	$URPSet_{30}$
$pms0021$, $pms0036$	0	0.3163	$URPSet_{31}$
$pms0031$, $pms0036$	0	0.2690	$URPSet_{32}$
$pms0029$, $pms0036$	0	0.2674	$URPSet_{33}$
$pms0021$, $pms0028$	0	0.2519	$URPSet_{34}$

Table 8: Unique Required Permission Sets in Malware Dataset (*CONTACTS*)

Permission Set	Support		Permission Set ID
	Clean	*Malware*	
$pms0008$, $pms0030$	0	0.3269	$URPSet_{35}$
$pms0008$, $pms0021$	0	0.2894	$URPSet_{36}$
$pms0008$, $pms0031$	0	0.2649	$URPSet_{37}$
$pms0008$, $pms0029$	0	0.2429	$URPSet_{38}$
$pms0028$, $pms0036$	0	0.2413	$URPSet_{39}$
$pms0008$, $pms0028$	0	0.2282	$URPSet_{40}$
$pms0008$, $pms0013$	0	0.2250	$URPSet_{41}$
$pms0028$, $pms0029$	0	0.2128	$URPSet_{42}$

Table 9: Unique Required Permission Sets in Malware Dataset (*WRITE_EXTERNAL_STORAGE*)

Permission Set	Support		Permission Set ID
	Clean	*Malware*	
$pms0004$, $pms0006$, $pms0023$	0	0.4475	$URPSet_{43}$
$pms0004$, $pms0030$	0	0.3896	$URPSet_{44}$
$pms0004$, $pms0005$, $pms0020$	0	0.3106	$URPSet_{45}$
$pms0004$, $pms0021$	0	0.2462	$URPSet_{46}$
$pms0004$, $pms0020$, $pms0023$	0	0.2258	$URPSet_{47}$
$pms0004$, $pms0029$	0	0.2022	$URPSet_{48}$

Table 10: Unique Required Permission Sets in Malware Dataset (*READ_PHONE_STATE*)

Permission Set	Support		Permission Set ID
	Clean	*Malware*	
$pms0005$, $pms0013$	0	0.5453	$URPSet_{49}$
$pms0005$, $pms0031$	0	0.5094	$URPSet_{50}$
$pms0005$, $pms0021$	0	0.4190	$URPSet_{51}$
$pms0005$, $pms0029$	0	0.3798	$URPSet_{52}$
$pms0005$, $pms0008$	0	0.3538	$URPSet_{53}$
$pms0005$, $pms0036$	0	0.3350	$URPSet_{54}$
$pms0005$, $pms0028$	0	0.2934	$URPSet_{55}$
$pms0005$, $pms0012$	0	0.2560	$URPSet_{56}$

In Table 6 - 10, we present the permission patterns that can have an impact on the following actions: access location information, read/write/send and receive SMS, access to contact list, write to external storage and access to phone state.

4.2.2. Unique Used Permission (UUP) Patterns

In Table 11, we list the combinations of the *used* permissions that are unique to the malware dataset only. It can be noted that the `INTERNET` permission is included in the top 3 permission combinations, $UUPSet_1$ to $UUPSet_3$ and appears in over 40% of the malware samples. The combination of `INTERNET` and `READ_PHONE_STATE` permission is another frequent permission pattern, as depicted by $UUPSet_{21}$ and $UUPSet_{30}$.

Table 11: Unique Used Permission Sets in Malware Dataset

Permission Set	Support		Permission Set ID
	Clean	*Malware*	
*pms*0001, *pms*0005, *pms*0006, *pms*0007	0	0.5542	$UUPSet_1$
*pms*0001, *pms*0005, *pms*0011	0	0.4687	$UUPSet_2$
*pms*0001, *pms*0006, *pms*0011	0	0.4320	$UUPSet_3$
*pms*0005, *pms*0006, *pms*0011	0	0.4312	$UUPSet_4$
*pms*0001, *pms*0007, *pms*0011	0	0.4149	$UUPSet_5$
*pms*0005, *pms*0007, *pms*0011	0	0.4133	$UUPSet_6$
*pms*0006, *pms*0007, *pms*0011	0	0.3855	$UUPSet_7$
*pms*0001, *pms*0002, *pms*0005, *pms*0007	0	0.3423	$UUPSet_8$
*pms*0001, *pms*0021	0	0.3358	$UUPSet_9$
*pms*0005, *pms*0021	0	0.3236	$UUPSet_{10}$
*pms*0001, *pms*0002, *pms*0011	0	0.2845	$UUPSet_{11}$
*pms*0002, *pms*0005, *pms*0011	0	0.2845	$UUPSet_{12}$
*pms*0001, *pms*0002, *pms*0006, *pms*0007	0	0.2829	$UUPSet_{13}$
*pms*0002, *pms*0005, *pms*0006, *pms*0007	0	0.2829	$UUPSet_{14}$
*pms*0002, *pms*0006, *pms*0011	0	0.2755	$UUPSet_{15}$
*pms*0007, *pms*0021	0	0.2723	$UUPSet_{16}$
*pms*0001, *pms*0020	0	0.2600	$UUPSet_{17}$
*pms*0005, *pms*0020	0	0.2600	$UUPSet_{18}$
*pms*0002, *pms*0007, *pms*0011	0	0.2568	$UUPSet_{19}$
*pms*0006, *pms*0020	0	0.2511	$UUPSet_{20}$
*pms*0001, *pms*0005, *pms*0010	0	0.2421	$UUPSet_{21}$
*pms*0001, *pms*0007, *pms*0010	0	0.2413	$UUPSet_{22}$
*pms*0005, *pms*0007, *pms*0010	0	0.2413	$UUPSet_{23}$
*pms*0011, *pms*0020	0	0.2380	$UUPSet_{24}$
*pms*0001, *pms*0006, *pms*0010	0	0.2372	$UUPSet_{25}$
*pms*0005, *pms*0006, *pms*0010	0	0.2372	$UUPSet_{26}$
*pms*0006, *pms*0007, *pms*0010	0	0.2364	$UUPSet_{27}$
*pms*0006, *pms*0021	0	0.2348	$UUPSet_{28}$
*pms*0007, *pms*0020	0	0.2266	$UUPSet_{29}$
*pms*0001, *pms*0003, *pms*0005, *pms*0007	0	0.2258	$UUPSet_{30}$
*pms*0002, *pms*0020	0	0.2185	$UUPSet_{31}$

Interestingly, `READ_LOGS` (*pms*0011) permission appears in one-third of

the permission patterns presented in Table 11. It is often combined with the `INTERNET` ($pms0001$) and `ACCESS_FINE_LOCATION` ($pms0002$) permissions. The remaining patterns include combinations of network-related and SMS permissions.

4.2.3. Common Required Permission (CRP) Patterns

Previously, we presented the permission patterns that were unique to malicious applications only. In Table 12, we list the permission combinations that appear in both clean and malware datasets. However, it can be observed based on the support value difference that the permission patterns are more prevalent in the malware dataset, as shown by the negative support difference values.

Table 12: Common Required Permission Sets in Both Clean and Malware Datasets

Permission Set	Support		Difference	Permission Set ID
	Clean	Malware		
$pms0001$, $pms0005$	0.3121	0.9307	−0.6186	$CRPSet_1$
$pms0005$	0.3187	0.9340	−0.6153	$CRPSet_2$
$pms0005$, $pms0023$	0.0236	0.6308	−0.6072	$CRPSet_3$
$pms0030$	0.0147	0.6210	−0.6064	$CRPSet_4$
$pms0001$, $pms0023$	0.0505	0.6349	−0.5844	$CRPSet_5$
$pms0023$	0.0522	0.6349	−0.5827	$CRPSet_6$
$pms0006$, $pms0023$	0.0399	0.6031	−0.5632	$CRPSet_7$
$pms0005$, $pms0006$	0.2421	0.7905	−0.5485	$CRPSet_8$
$pms0001$, $pms0005$, $pms0006$	0.2421	0.7897	−0.5477	$CRPSet_9$
$pms0001$, $pms0004$, $pms0005$	0.1328	0.6544	−0.5216	$CRPSet_{10}$
$pms0004$, $pms0005$	0.1337	0.6553	−0.5216	$CRPSet_{11}$
$pms0013$	0.0489	0.5542	−0.5053	$CRPSet_{12}$
$pms0031$	0.0106	0.5159	−0.5053	$CRPSet_{13}$
$pms0001$, $pms0004$, $pms0005$, $pms0006$	0.1149	0.5623	−0.4474	$CRPSet_{14}$
$pms0004$, $pms0005$, $pms0006$	0.1149	0.5623	−0.4474	$CRPSet_{15}$
$pms0004$, $pms0023$	0.0293	0.4637	−0.4344	$CRPSet_{16}$
$pms0021$	0.0277	0.4417	−0.4140	$CRPSet_{17}$

Permissions, such as `INTERNET` ($pms0001$), `READ_PHONE_STATE` ($pms0005$), `ACCESS_NETWORK_STATE` ($pms0006$) and `ACCESS_WIFI_STATE` ($pms0023$), are present in different combinations and appear in more than 40% of the malware dataset. One interesting observation is $CRPSet_{14}$ which comprises of a combination of four permissions and appear in a significant 40% of the malicious applications.

4.2.4. Common Used Permission (CUP) Patterns

In Table 13, we present the *used* permission combinations that appeared in both the clean and malware datasets. We note that although both

datasets have the same permission patterns, the ones in the malware dataset have higher support values.

In such patterns, only five permissions are found: `INTERNET` ($pms0001$), `READ_PHONE_STATE` ($pms0005$), `ACCESS_NETWORK_STATE` ($pms0006$), `VIBRATE` ($pms0007$) and `READ_LOGS` ($pms0011$). It is also worth noting that $CUPSet_1$ and $CUPSet_2$ have almost the same support difference, hence indicating that the occurrence of these permission combinations are highly relevant. Moreover, we observed that even though `READ_LOGS` ($pms0011$) permission did not appear in the common *required* permission patterns, but it appeared in three common unique permission patterns `READ_LOGS`, $CUPSet_7$ - $CUPSet_9$.

Table 13: Common Used Permission Sets in Both Clean and Malware Datasets

Permission Set	Support		Difference	Permission Set ID
	Clean	*Malware*		
$pms0001$, $pms0005$	0.2991	0.9152	−0.6161	$CUPSet_1$
$pms0005$	0.3032	0.9169	−0.6137	$CUPSet_2$
$pms0001$, $pms0005$, $pms0006$	0.2363	0.7718	−0.5355	$CUPSet_3$
$pms0005$, $pms0006$	0.2363	0.7718	−0.5355	$CUPSet_4$
$pms0001$, $pms0005$, $pms0007$	0.2168	0.6512	−0.4344	$CUPSet_5$
$pms0005$, $pms0007$	0.2192	0.6528	−0.4336	$CUPSet_6$
$pms0005$, $pms0011$	0.0538	0.4686	−0.4148	$CUPSet_7$
$pms0011$	0.0693	0.4760	−0.4067	$CUPSet_8$
$pms0001$, $pms0011$	0.0685	0.4711	−0.4026	$CUPSet_9$

4.3. Discussion

The *Android* smartphone has gained in popularity in the past few years. Two main factors that contributed towards this change is the open-source nature of the platform and the flexibility provided to users and developers alike when downloading and developing *Android* applications, respectively. However, not all applications present on the application markets, both official and third-party, are clean. Previous work showed that malware authors take advantage of the *Android* permission system to entice users into installing unsafe applications. As such, this study aims to understand *required* and *used* permissions by *Android* applications by applying data mining techniques to find emerging permission patterns that can be used to contrast clean and malicious applications.

4.3.1. Observations from Statistical Analysis

Our proposed methodology considers the patterns of both *required* and *used* permissions. From our statistical analysis in Section 3.1, we observe that the `INTERNET` permission remain the most *required* (97.72%) and *used* (94.62%) permission in our experimental dataset. We also find, from Tables 1 and 2, that there is a significant difference in the frequencies of *required* and *used* permissions for the clean and malware datasets. This observation aligns with the one made by Felt et al. in [34] and therefore, demonstrates that both clean and malicious applications can be over-privileged. Till date, most of the proposed solutions have only considered *required* permissions which are extracted from the *AndroidManifest.xml* files. From our statistical results, we argue that *used* permissions should also be considered as part of the feature set and as such, can aid towards malicious application detection.

Additionally, in order to have a comparative distribution of *required* and *used* permissions, we extend the statistical analysis by applying the *biclustering* algorithm to generate visualization maps. It should be noted that we apply *biclustering* mainly to visualize the distribution of *required* and *used* permissions. Thus, we do not aim to identify clusters of permissions. As expected, since applications generally request more permissions than actually used, the distribution of *required* permissions for clean and malware datasets is more sparsed than that of *used* permissions - as shown in Figures 2 and 3, respectively. The visualization maps provide researchers and analysts alike with a first-hand overview of permissions that are common and unique between clean and malicious applications. Furthermore, the maps can be used as a substitution for statistical analysis as it is a time-consuming process and requires little or no margin of error. As Zhou et al. pointed out in [45], the increasing number of malicious applications is mostly due to how easy it is to produce repackaged applications. These applications can contain additional advertising libraries, malicious code and most importantly, additional permissions that were previously not present in the original applications. The maps can outline these differences in permissions for clean and malicious applications. Subsequently, the permission distribution visuals can also portray *required* and *used* permissions for different variants belonging to the same malware family. Hence, the maps can be used for a preliminary analysis of zero-day samples detected by antivirus companies.

4.3.2. Analysis of Permission Visualizations

From the *biclustering* results, we observed that several applications, clean or malicious, have more than one (*required* and *used*) permission in

common between them. Conversely, we also noticed similar observation for unique *required* and *used* permissions for the two datasets. In general, existing work [12], [30], [33] consider only individual permissions when studying permission request patterns. Therefore, we put forward a method that consider co-dependencies between permissions that are unique and common amongst clean and malicious applications. In our paper, we apply a data mining technique known as contrast mining to generate permission sets that constitute of multiple permissions and can be used to reinforce similarities and contrasts between clean and malicious applications.

From our findings, we observe that 23 permissions (as shown in Table 4), out of a total of 128 permissions that were extracted from our dataset, appear frequently in the permission sets. It is also worth noting that two of the *Dangerous* permissions, `WRITE_EXTERNAL_STORAGE` (*pms*0004) and `READ_PHONE_STATE` (*pms*0005) can be implicitly granted to an application that utilizes API level 3 or lower, as described in [27]. This implies that if these two permissions are not recorded as *required* permissions, they can still be present as *used* permissions. Upon further investigation, we found that whilst the number of *required* permission for `WRITE_EXTERNAL_STORAGE` exceeds that of *used* permission, the same observation cannot be made for `READ_PHONE_STATE`. From Tables 1 and 2, it can be noticed that the number of clean applications (391) which *required* `READ_PHONE_STATE` is lesser than the number of clean applications (457) that *used* this permission. Although we do not keep record of the API level information for the applications in our dataset; however, in this case, we can deduce that some clean applications from the set of 457 applications were implicitly granted the `READ_PHONE_STATE` permission. This permission can have nefarious ramifications on users' private information as it allows an application to read unique device identifiers such as, *International Mobile Equipment Identity* (IMEI), *International Mobile Subscriber Identity* (IMSI) and the *SIM* serial number, as shown by [46].

4.3.3. Analysis of Contrast Permission Patterns

In Section 4.2, we present the most significant permission sets generated by contrast mining. Based on our experimental results, we found that a large number of *required* and *used* permission sets were unique in malicious applications only. This is a good indication that the permission sets can be further applied during the malware detection phase to identify malicious applications. For *normal* required permissions, we observed from Table 5 that the permission set IDs, $URPSet_1$ and $URPSet_2$ were required by 63% and 60% of the malicious applications in our dataset, respectively. We deduce

that this might be the case due to the fact that 25% of our experimental malware samples (malicious applications) belong to the *DroidKungFu3* malware family. As demonstrated in [47], malware samples classified under *DroidKungFu3* attempt to extract device ID, network-related information and send all information back to the attacker's server.

As for the *Dangerous* required permissions sets included in Tables 6 to 10, we notice several interesting permission sets on which we provide further explanation. For permission set IDs $URPSet_{26}$ and $URPSet_{27}$, we find that 25% of malicious applications require both `ACCESS_FINE_LOCATION` ($pms0002$) and `ACCESS_COARSE_LOCATION` ($pms0020$) permissions. While `ACCESS_COARSE_LOCATION` grants access to GPS location sources, `ACCESS_COARSE_LOCATION` is used for location information related to network sources. However, the documentation [48] provided by *Google* specifies that if a developer requires network and GPS location information, they do not need to include both permissions in the application; only requesting `ACCESS_FINE_LOCATION` should suffice. The presence of unused permission can be exploited via permission inheritance during inter-component communications, as explained in [49].

Moreover, we observe that for SMS-related permissions: `SEND_SMS` ($pms0021$), `RECEIVE_SMS` ($pms0029$), `READ_SMS` ($pms0030$) and `WRITE_SMS` ($pms0031$), as shown in Table 7, the `CALL_PHONE` ($pms0036$) permission is associated with these four permissions in over 25% of the malicious applications in our dataset. The `CALL_PHONE` permission allows an application to proceed with making a phone call without going through the usual dialer interface. Some malware samples exploit the aforementioned permission combinations to make premium calls and send text messages to premium numbers.

As for the *used* permission sets that are unique in our malware dataset (Table 11), we observed that the permission set: `INTERNET` ($pms0001$), `READ_PHONE_STATE` ($pms0005$), `ACCESS_NETWORK_STATE` ($pms0006$), `VIBRATE` ($pms0007$) with permission set ID $UUPSet_1$ was used by 55% of the malware samples. Interestingly, we also found that the same permission set was present in Table 5 under the permission set ID $URPSet_{11}$, with the only exception that it was required by only 35% of the malware samples. We attribute this 20% difference to the observation made in Section 4.3.2, on the `READ_PHONE_STATE` permission. Moreover, it can be noted from Table 11 that the `READ_LOGS` ($pms0011$) permission is frequently associated with the permission sets and appeared in 25% to 50% of the malware dataset. There was previously no indication that ($pms0011$) was a highly *used* permission among malicious applications as the permission did not appear in the Top 20 most *used* permission, in Table 2. This further consolidates our argument that permission patterns cannot be generated by only considering the

number of frequencies for that particular permission.

Furthermore, we also noted that there are several permission sets which appeared in both clean and malware datasets, shown in Tables 12 and 13. The negative support difference given in the table shows that the permission sets are more prevalent in malicious applications than in clean ones. We observed that the top two permission sets, $CRPSet_1$ and $CRPSet_2$ in Table 12and $CUPSet_1$ and $CUPSet_{11}$ in Table 13 are the same. However, we noted some discrepancies for permissions such as `READ_LOGS` ($pms0011$) and `VIBRATE` ($pms0007$). Similar to our previous observations, it appears that the above two permissions are not recorded during the generation of *required* permission sets but for *used* permission sets, their high support values indicate that they are highly significant.

5. Conclusion

In this paper, we studied the *Android* permission system as the smartphone platform makes use of permissions to regulate access to system resources and users' private information. In order to understand and identify permission patterns, the existing work consider only those permissions that are declared in the *AndroidManifest.xml* files. We refer to those permissions as *required* permissions. However, there is another permission check that takes place after an application has been installed and is executed by the smartphone OS. We refer to such permissions as *used* permissions.

In our work, we considered the implications of incorporating *used* permissions in permission patterns and determined their usefulness in contrasting between clean and malicious applications. We proposed an efficient pattern mining method to generate a set of emerging contrast permission patterns for our clean and malware dataset. Based on our experimental results, we observed that there are several permissions that do not appear in the *required* permission sets but are present in the *used* permission sets. We found out that there is a discrepancy in the official documentation that allows application with API level 3 or level to implicitly inherit certain permissions - although they are not declared in the *AndroidManifest.xml* file.

Additionally, the patterns obtained from our proposed methodology ensures that the permission sets were not generated by chance as we used support values to measure and rank the patterns. This is an improvement over Frank et al.'s work [12] where the authors had to simulate permission request data to test their generated patterns. Last but not least, since obfuscation methods cannot be applied to *Android* permissions, the generated permission sets can be used to contrast clean and malicious applications. In

the future, we would like to work on finding contrasting patterns that can differentiate between an original application and a repackaged one.

References

[1] PC Magazine. Encyclopedia, Accessed in December 2012. http://www.pcmag.com/encyclopedia_term/0,2542,t=Smartphone&i=51537,00.asp.

[2] Google. Nexus, Accessed in March 2013. http://www.google.com/nexus.

[3] Apple Inc. iphone, Accessed in March 2013. http://www.apple.com/iphone.

[4] Research in Motion Ltd. Blackberry, Accessed in March 2013. http://au.blackberry.com.

[5] Microsoft. Windows phone, Accessed in March 2013. http://www.windowsphone.com/en-gb.

[6] Apple Inc. Welcome to apple store, Accessed in March 2013. http://store.apple.com/au.

[7] BlackBerry. Blackberry world, Accessed in March 2013. http://appworld.blackberry.com/webstore.

[8] Google. Google play, Accessed in March 2013. https://play.google.com/store.

[9] P. Gilbert and C. Byung-Gon and P. C. Landon and J. Jaeyeon. Vision: automated security validation of mobile apps at app markets. In *Proceedings of the 2nd International Workshop on Mobile Cloud Computing and Services (MCS 2011)*, pages 21–26, Washington, USA, June 2011.

[10] Google. Malware - what's the policy?, Accessed in March 2013. http://support.google.com/adwordspolicy/bin/answer.py?hl=en&answer=1308246&topic=1626336&path=1316546&ctx=leftnav.

[11] Y. Zhou and X. Jiang. Dissecting Android malware: Characterization and evolution. In *Proceedings of the IEEE Symposium on Security and Privacy (SP 2012)*, pages 95–109, San Francisco, CA, May 2012.

[12] M. Frank, B. Dong, A.P. Felt, and D. Song. Mining permission request patterns from Android andFacebook applications. In *Proceedings of the IEEE International Conference on Data Mining (ICDM 2012)*, pages 1–16, Brussels, Belgium, December 2012.

[13] D. Barrera, H.G. Kayacik, P.C. van Oorschot, and A. Somayaji. A methodology for empirical analysis of permission-based security models and its application to Android. In *Proceedings of the 17th ACM conference on Computer and communications security (CCS 2010)*, pages 73–84, Chicago, Illinois, USA, October 2010.

[14] A.P. Felt, K. Greenwood, and D. Wagner. The effectiveness of application permissions. In *Proceedings of the USENIX Conference on Web Application Development (WebApps 2011)*, pages 1–12, Portland, Oregon, June 2011.

[15] A.P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: User attention, comprehension and behavior. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS 2012)*, number 3, pages 1–14, Washington, D.C., July 2012.

[16] P.H. Chia, Y. Yamamoto, and N. Asokan. Is this app safe? a large scale study on application permissions and risk signals. In *Proceedings of the 21st international conference on World Wide Web (WWW 2012)*, pages 311–320, Lyon, France, April 2012.

[17] International Secure Systems Lab. Andrubis: Analyzing Android binaries, Accessed in May 2012. `http://anubis.iseclab.org/?action=home`.

[18] virusTotal. Credits & acknowlegements, Accessed in March 2013. `https://www.virustotal.com/en/about/credits`.

[19] Open Handset Alliance. Android, Accessed in February 2013. `http://www.openhandsetalliance.com/android_overview.html`.

[20] F. Ableson. Introduction to Android development, Accessed in January 2013. `http://www.ibm.com/developerworks/library/os-android-devel`.

[21] Google. Android SDK, Accessed in January 2013. `http://developer.android.com/sdk/index.html`.

[22] The Eclipse Foundation. Eclipse ide for java developers, Accessed in March 2013. http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/junosr1.

[23] Android Open Source Project. Bytecode for the dalvik virtual machine, Accessed in January 2013. http://source.android.com/tech/dalvik/dalvik-bytecode.html.

[24] Google. Google play, Accessed in March 2013. https://play.google.com.

[25] H. Lockheimer. Android and Security, Accessed in March 2013. http://googlemobile.blogspot.com.au/2012/02/android-and-security.html.

[26] J.R. Raphael. Inside Android 4.2's powerful new security system, Accessed in November 2012. http://blogs.computerworld.com/android/21259/android-42-security.

[27] Google. Android permissions, Accessed in December 2012. http://developer.android.com/guide/topics/manifest/permission-element.html.

[28] A. Bartel, J. Klein, M. Monperrus, and Y.L. Traon. Automatically securing permission-based software by reducing theattack surface - an application to Android. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 2012)*, pages 274–277, Essen, Germany, September 2012.

[29] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P.G. Bringas, and G. Álvarez. PUMA: Permission usage to detect malware in Android. In *Proceedings of the International Joint Conference CISIS'12-ICEUTE'12-SOCO'12 Special Sessions*, volume 189 of *Advances in Intelligent Systems and Computing*, pages 289–298. Springer Berlin Heidelberg, September 2013.

[30] I. Rassameeroj and Y. Tanahashi. Various approaches in analyzing Android applications with its permission-based security models. In *Proceedings of the IEEE International Conference on Electro/Information Technology (EIT 2011)*, number 44, pages 1–6, Minnesota, USA, May 2011.

[31] J. Sahs and L. Khan. A machine learning approach to Android malware detection. In *Proceedings of the European Intelligence and Security Informatics Conference (EISIC 2012)*, pages 141–147, Odense, Denmark, August 2012.

[32] D.J. Wu, C.H. Mao, T.E. Wei, H.M. Lee, and K.P. Wu. DroidMat: Android malware detection through manifest and API calls tracing. In *Proceedings of the 2012 Seventh Asia Joint Conference on Information-Security (Asia JCIS 2012)*, pages 62–69, Tokyo, Japan, August 2012.

[33] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang. Hey, You, Get Off of My Market: Detecting Malicious Appsin Official and Alternative Android Markets. In *In Proceedings of the 19th Annual Network and Distributed System Security Symposium (NDSS 2012)*, pages 1–13, San Diego, California, February 2012.

[34] A.P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In *Proceedings of the ACM Conference and Communications Security (CCS 2011)*, pages 627–638, Chicago, U.S.A, October 2011.

[35] T. Vidas, N. Christin, and L. Cranor. Curbing Android permission creep. In *Proceedings of the 2011 Web 2.0 Security and Privacy Workshop (W2SP 2011)*, pages 1–5, Oakland, CA, May 2011.

[36] A.P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner. A survey of mobile malware in the wild. In *Proceedings of the ACM Workshop on Security and Privacy in Mobile Devices (SPSM 2011)*, pages 3–14, Chicago, U.S.A, October 2011.

[37] S.C. Madeira and A.L. Oliveira. Biclustering algorithms for biological data analysis: A survey. *IEEE Transactions on computational Biology and Bioinformatics*, 1(1):24–45, 2004.

[38] G.J. Szekely and M.L. Rizzo. Hierarchical clustering via joint between-within distances: Extending ward's minimum variance method. *Journal of Classification*, 22(2):151–183, 2005.

[39] A. Fernández and S. Gómez. Solving non-uniqueness in agglomerative hierarchical clustering using multidendrograms. *Journal of Classification*, 25(1):43–65, 2008.

[40] Joe H. Ward. Hierarchical grouping to optimize an objective function. *Journal of The American Statistical Association*, 58:236–244, 1963.

[41] R. Agrawal, T. Imieinski, and A. Swami. Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia, editors, *Proceedings of the ACM SIGMOD International Conference on the Management of Data (COMAD 1993)*, pages 207–216, Washington DC, 1993. ACM Press.

[42] S. Liu, R. Law, J. Rong, G. Li, and J. Hall. Analyzing changes in hotel customers' expectations by trip mode. *International Journal of Hospitality Management*, 34:359–371, 2013.

[43] J. Rong, H.Q. Vu, R. Law, and G. Li. A behavioral analysis of web sharers and browsers inhong kong using targeted association rule mining. *Tourism Management*, 33(4):731–740, 2012.

[44] R. Law, R. Rong, H.Q. Vu, G. Li, and H.A Lee. Identifying changes and trends in hong kong outbound tourism. *Tourism Management*, 32(5):1106–1114, 2011.

[45] W. Zhou, Y. Zhou, X. Jiang, and P. Ning. Detecting repackaged smartphone applications in third-party Android marketplaces. In *Proceedings of the second ACM conference on Data and ApplicationSecurity and Privacy (CODASPY 2012)*, pages 317–326, San Antonio, Texas, USA, February 2012.

[46] V. Moonsamy, M. Alazab, and L. Batten. Towards an understanding of the impact of advertising on data leaks. *International Journal of Security and Networks*, 7(3):181–193, 2012.

[47] F-Secure. Trojan:android/droidkungfu.c, Accessed in January 2013. `http://www.f-secure.com/v-descs/trojan_android_droidkungfu_c.shtml`.

[48] Android Developer. Location strategies, Accessed in January 2013. `http://developer.android.com/guide/topics/location/strategies.html`.

[49] E. Chin, A.P. Felt, K. Greenwood, and D. Wagner. Analyzing inter-application communication in Android. In *Proceedings of the 9th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys 2011)*, pages 239–252, Washington, USA, June 2011.