Contrasting Permission Patterns between Clean and Malicious Android Applications

Veelasha Moonsamy, Jia Rong, Shaowu Liu, Gang Li, and Lynn Batten

School of Information Technology, Deakin University
221 Burwood Highway, VIC 3125, Australia
v.moonsamy@research.deakin.edu.au, jiarong@acm.org,
{swliu,gang.li,lynn.batten}@deakin.edu.au

Abstract. The *Android* platform uses a permission system model to allow users and developers to regulate access to private information and system resources required by applications. Permissions have been proved to be useful for inferring behaviors and characteristics of an application. In this paper, a novel method to extract contrasting permission patterns for clean and malicious applications is proposed. Contrary to existing work, both *required* and *used* permissions were considered when discovering the patterns. We evaluated our methodology on a clean and a malware dataset, each comprising of 1227 applications. Our empirical results suggest that our permission patterns can capture key differences between clean and malicious applications, which can assist in characterizing these two types of applications.

Keywords: Android Permission, Malware Detection, Contrast Mining, Permission Pattern.

1 Introduction

The increase in *Android* smartphone sales has led to a surge in the number of applications available on application markets. Additionally, the freedom of installing applications from third-party markets, rather than being constrained to only the official market, has boosted the number of *Android* applications. This, in turn, has incentivized application developers to churn out applications and upload them on different third-party markets [1]. As no application review process is in place for third-party markets, the cleanliness of these applications cannot be guaranteed [2]. Users can only rely on the description and permissions listed on the application market to decide whether or not they should install an application.

Android platform employs a permission system to restrict application privileges in order to secure a user's private information [3]. However, its effectiveness highly depends on the user's comprehension of permission approval [4]. The permissions requested during application installation are referred to as *required permissions*. Unfortunately, as noted by Felt et al. [4], not all the users read or understand the warnings of *required* permissions shown during installation. In

T. Zia et al. (Eds.): SecureComm 2013, LNICST 127, pp. 69–85, 2013.
© Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2013

order to have a better understanding of permission requests, Frank et al. [3] proposed a probability model to identify the common *required* permission patterns for all *Android* applications. Zhou and Jiang [5] listed the top *required* permissions for both clean and malicious applications, but only individual permissions were considered by frequency counting.

We observed that the following issues have been overlooked in the area of *Android* permissions analysis:

- *Contrasting Permissions Patterns.* Despite the numerous research endeavors [4,6,7] aimed at interpreting *Android* permissions and their combinations, there is no existing work that aims at identifying the permission differences between clean and malicious *Android* applications.
- *Used Permission.* No work has considered incorporating *used* permissions, which can be extracted from static analysis by the *Andrubis* system [8], into the permission patterns. Compared to *required* permissions, *used* permissions provide a better understanding of the permissions that are needed by an application in order to function properly. Whenever an API call is invoked during the execution of an application, the *Android* platform will verify if the API call is permission-protected before proceeding to execute the call; such permissions are referred to as *used* permissions.

With the availability of the *Andrubis* framework and the advances in data mining, it is now possible to consider both *required* and *used* permissions, together with the use of our new pattern mining algorithm to generate contrasting permission patterns for clean and malicious applications. While most of the existing work is based on *required* permissions, *used* permissions are equally important and should be considered to better differentiate between permission patterns for clean and malicious applications. Therefore, our aim is *to identify a set of unique and common permission patterns that can contrast clean applications from malicious ones.*

In order to apply our pattern mining technique to identify the desired contrast permission patterns, a clean and a malware dataset are considered. In 2012, Zhou and Jiang [5] published the first benchmark dataset of malicious applications, which comprises of 49 malware families. The applications were collected from third-party markets between August 2010 and October 2011. As there was no clean dataset publicly available, we proceeded to collect our own clean applications that were released during the same time period as the malware dataset. The clean applications were downloaded from two popular third-party application markets: *SlideME* (http://slideme.org) and *Pandaapp* (http://android.pandaapp.com). The applications were sorted based on the number of downloads and the ratings given by the users, and only the top ones were selected.

To our knowledge, this work reports one of the first pattern mining methods that can generate unique and common permission patterns, which include both *required* and *used* permissions, for clean and malicious applications. The novelty and contributions of this work can be summarized as follows:

- To find the permission combinations, a new contrast permission pattern mining algorithm (CPPM) is proposed to identify the permission patterns that can significantly differentiate between clean and malicious applications.
- To our knowledge, this is the first work to incorporate both *required* and *used* permissions to generate permission patterns. Based on our empirical results, it can be deduced that such patterns can help to contrast clean applications from malicious ones.

The rest of the paper is organized as follows: Section 2 briefly reviews the *Android* platform, 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 methods followed by the proposed contrast pattern mining algorithm. The experiments and the obtained results are then reported in Section 4 together with a discussion of our findings. Finally, Section 5 concludes the paper together with our future work.

2 Background and Related Work

2.1 Android and Its Permission System

Android is a Linux-based Operating System (OS) which was designed and developed by the *Open Handset Alliance* in 2007 [9]. The *Android* platform is made up of multiple layers consisting of the Linux-kernel, libraries and an application framework with built-in applications [10]. Additional applications can be downloaded and installed from either official market, *Google Play* [11], or third-party markets.

Google applies the permission system as a measure to restrict access to privileged system resources. Developers have to explicitly mention the permissions, that require 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 they are classified into four categories: *Normal*, *Dangerous*, *Signature* and *SignatureOrSystem* [12].

- *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 Application Programming Interfaces (APIs) are 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 privacy-relevant resources from any unauthorized operations. However, it cannot deter malware developers from declaring additional *required* permissions for their applications. From the above observation, several studies [3, 6, 7] have tried to identify the common *required* permissions that are frequently declared by *Android* application developers.

2.2 Android Permissions and Related Work

Understanding Android Permissions. Frank et al. [3] selected 188,389 applications from the official market and analyzed the combinations of permission requests by these applications using a probabilistic model. Bartel et al. [13] 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, ensured that the user did not grant access to unnecessary permissions when installing the application. A model designed by Sanz et al. [14] was based on features that comprised solely of *Android* permissions, which helped to understand the *Android* permission system and the patterns for *normal* permission requests.

Permission-Based Malware Detection. Malware detection is an emerging topic in the study of the *Android* platform with many successful achievements; however, not much attention has been paid on detection using permission patterns. Chia et al. [7] argued that the current user-rating system is not a reliable source of measurement to predict whether or not an application is 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 [15] focused on feature representation as one of the challenges to malware detection. The features 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 the authors applied a one-class *Support Vector Machine* to train the classifiers. However, the evaluation results showed that the common features existing in both the clean and malware datasets affected the detection error rate.

Wu et al. [16] put forward a static feature-based technique that can aid towards malware detection. First, they applied *K-means* algorithm to generate the clusters and used *Singular Value Decomposition* to determine the number of clusters. In the second step, they classified clean and malicious applications using the *k-Nearest Neighbor* (kNN) algorithm.

Zhou et al. [17] proposed a two-layered system, known as *DroidRanger* and used "permission-based behavioral foot-printing and heuristics-based filtering". The authors observed that the permissions extracted from the malicious applications gave an insight into uncommon permission requests by some malware families.

In [14], Sanz et al. proposed to extract the permissions and the hardware features to build the feature set. As a result, they observed that clean applications required two to three permissions on average, but some of malicious applications only had one permission and were still able to carry out the attack.

2.3 Summary and Problem Identification

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. However, there is no evidence providing a clear understanding on the key differences for permissions in clean and malicious applications.

Thus, we identify the following research questions:

- How can we measure the similarities and differences between permission requests for clean and malicious applications?
- What method can be used to incorporate *used* permissions in the permission patterns?

To answer these questions, we have extended the current statistical method used for identifying both *required* and *used* permission patterns in *Android* applications. A contrast pattern mining technique has been proposed to identify the most useful permission combinations that can distinguish between clean and malicious applications.

3 Mining Contrast Permission Patterns

3.1 Experimental Dataset

For our malware dataset, we used Zhou and Jiang's [5] collection of 1227 malicious applications, which comprises of 49 malware families. These were collected from third-party markets between August 2010 and October 2011. In order to maintain the same timeline as the malware dataset, we proceeded to collect our set of 1227 clean applications that were released during the same period as the malicious ones. The clean applications were downloaded from two popular third-party application markets: *SlideME* (http://slideme.org) and *Pandaapp* (http://android.pandaapp.com). The applications were sorted based on the number of downloads and the ratings given by the users, and only the top ones were selected.

3.2 Statistical Analysis on Android Permissions

Statistical analysis has been widely used to analyze *Android* permissions. Accordingly, we started our work with an initial analysis on the clean and malware datasets using frequency counting and extended Zhou and Jiang's work [5] to explore *used* permissions. A novel contrast pattern mining algorithm is then presented to identify specific permission patterns that differentiate clean applications from malicious ones.

We employed statistical analysis to study both *required* and *used* permissions for clean applications as well as malicious ones. Based on the aforementioned two types of permissions for clean and malicious applications, we further generated the following four sub-datasets: (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 was employed on all four sub-datasets to find out the most popular permissions required or used.

By comparing the top 20 *required* permissions for clean and malicious applications listed in Table 1, we found that malicious applications requested a total of 14,758 permissions, in contrast to the 4,470 permissions requested by clean applications. Among these permissions, we found some of them only appeared in one dataset, in other words, those permissions were only *required* or *used* by clean applications but not malicious ones, and vice versa. We refer to these permissions as the *'unique permissions'*. Similarly, we name those permissions that appear in both clean and malware datasets the *'common permissions'*. In total, there are 33 unique *required* permissions for clean applications and 20 for malicious ones; and also 70 common *required* permissions. Another 5 permissions were never requested by any application. For *used* permissions, there are 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 was never used by any application. The four most frequently requested common permissions by both clean and malicious applications are: `INTERNET`, `ACCESS_COARSE_LOCATION`, `WRITE_EXTERNAL_STORAGE` and `VIBRATE`.

In contrast, among the top 20 *required* permissions, 9 of them appeared frequently in the malware dataset. Moreover, when comparing the top 20 *used* permissions in clean and malicious applications in Table 2, we observed that 16 out of 20 popular *used* permissions were common in both datasets.

Statistical analysis such as direct frequency counting is suitable for identifying single permissions that are popular in each sub-dataset. 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 extended the analysis of *Android* permissions by proposing a contrast pattern mining algorithm.

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

Clean Applications		Malicious Applications	
Required Permission	*Frequency*	*Required Permission*	*Frequency*
INTERNET	1121	INTERNET	1199
ACCESS_NETWORK_STATE	663	ACCESS_COARSE_LOCATION	1146
READ_PHONE_STATE	391	VIBRATE	994
WRITE_EXTERNAL_STORAGE	362	WRITE_EXTERNAL_STORAGE	823
ACCESS_COARSE_LOCATION	236	READ_SMS	779
VIBRATE	210	WRITE_SMS	762
WAKE_LOCK	188	READ_CONTACTS	680
ACCESS_FINE_LOCATION	162	BLUETOOTH	633
GET_TASKS	125	WRITE_CONTACTS	542
SET_WALLPAPER	102	DISABLE_KEYGUARD	491
ACCESS_WIFI_STATE	64	WAKE_LOCK	471
RECEIVE_BOOT_COMPLETED	60	RECORD_AUDIO	461
READ_CONTACTS	58	ACCESS_FINE_LOCATION	446
WRITE_SETTINGS	45	ACCESS_NETWORK_STATE	416
CAMERA	43	READ_PHONE_STATE	414
CALL_PHONE	42	SET_ORIENTATION	413
SEND_SMS	34	CHANGE_WIFI_STATE	384
RESTART_PACKAGES	32	READ_LOGS	361
RECEIVE_SMS	31	BLUETOOTH_ADMIN	342
RECORD_AUDIO	27	RECEIVE_BOOT_COMPLETED	325

3.3 Contrast Permission Pattern Mining

In order to discover a set of permission patterns that can visibly show contrast between clean and malicious applications, we propose the *Contrast Permission Pattern Mining* (CPPM) method. The output permission patterns were 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 individual and combined permissions and their combinations into consideration. Two major processes were involved in *CPPM*: (1) candidate permission itemset generation, and (2) contrast permission pattern selection, as illustrated in Fig. 1.

1. **Candidate Permission Itemset Generation**
   The purpose of this process is to obtain a number of candidate permission combinations that are likely to be the expected contrast patterns. *CPPM* takes at least two datasets as input. In our case two datasets were loaded, each of which contained either clean or malicious applications. We generated the candidate permission itemsets from every dataset using the same procedure, which included 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 applications. Let $I = \{A, B, C \dots\}$ be the set of possible items in Dx. Each item can be considered as a permission required or used by an application and an itemset is formed by a set of items (permissions required or used). The *Apriori-based* approach [18] 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

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

Clean Applications		Malicious Applications	
Used Permission	*Frequency*	*Used Permission*	*Frequency*
INTERNET	1029	INTERNET	1161
WAKE_LOCK	816	ACCESS_COARSE_LOCATION	1125
ACCESS_NETWORK_STATE	738	VIBRATE	954
VIBRATE	608	WAKE_LOCK	826
READ_PHONE_STATE	457	ACCESS_WIFI_STATE	584
ACCESS_COARSE_LOCATION	372	ACCESS_NETWORK_STATE	519
SET_WALLPAPER	126	READ_SMS	473
ACCESS_FINE_LOCATION	116	WRITE_CONTACTS	426
GET_ACCOUNTS	98	READ_PHONE_STATE	354
ACCESS_WIFI_STATE	85	RECORD_AUDIO	319
READ_SMS	82	SET_WALLPAPER	297
RESTART_PACKAGES	65	ACCESS_FINE_LOCATION	199
GET_TASKS	61	GET_ACCOUNTS	178
CHANGE_CONFIGURATION	55	GET_TASKS	124
RECEIVE_SMS	37	RECEIVE_BOOT_COMPLETED	111
FLASHLIGHT	37	ACCESS_CACHE_FILESYSTEM	101
WRITE_CONTACTS	34	WRTIE_OWNER_DATA	59
RECEIVE_BOOT_COMPLETED	23	CHANGE_CONFIGURATION	52
WRTIE_OWNER_DATA	12	READ_HISTORY_BOOKMARKS	49
WRITE_SETTINGS	10	EXPAND_STATUS_BAR	41

each iteration, one new item is tentatively added into the existing candidate itemset. However, the *Apriori-based* approach can generate a large number of candidate itemsets with high computational cost. To alleviate this problem, a support-based pruning technique is employed 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:

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

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. However, in our case, the statistical analysis results showed most of the unique permissions were requested or used by few applications. This indicated that they have low support value. In order to inadvertently miss any valuable patterns, we decided to take both frequent and infrequent candidate itemsets, but only used frequent ones to generate new candidate itemsets to cut down the computational cost.

2. **Contrast Permission Pattern Selection**
The permission itemsets obtained from the previous steps 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 contrasts are shown by the different occurrence behaviors

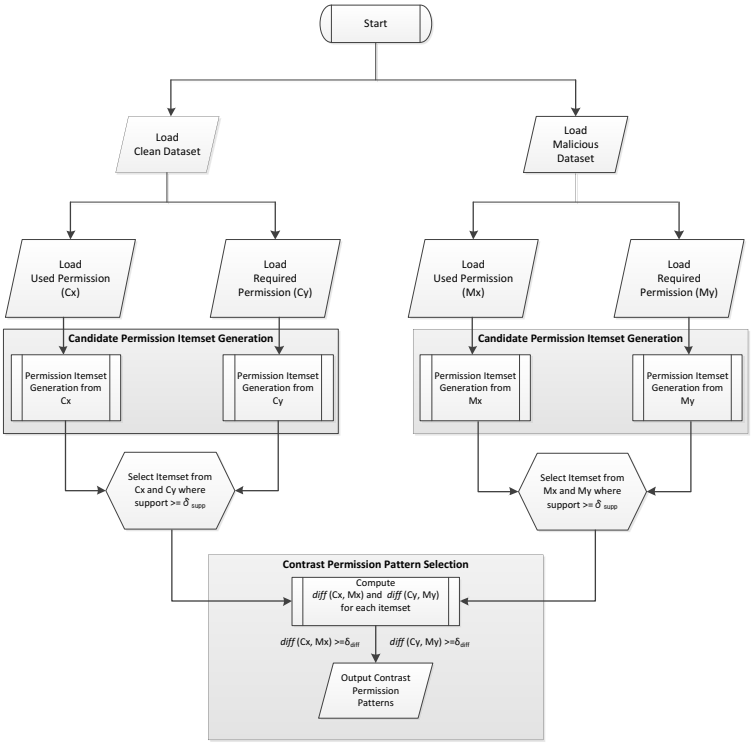

Fig. 1. CPPM-based Framework

in 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 supports between two datasets. The bigger the difference is in support values, the greater the contrast a permission pattern has.

Given one candidate permission itemset $\{A, B\}$ and its *supports* in clean and malware datasets, $supp(A, B)_{clean}$ and $supp(A, B)_{malicious}$, calculate the difference by $diff(A, B) = supp(A, B)_{clean} - supp(A, B)_{malicious}$. Then, $\{A, B\}$ 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 not all the permissions were required or used. Hence, to evaluate the proposed CPPM algorithm, we ignored the permissions

Table 3. Four Sub-datasets Used in CPPM Experiments

	Dataset	Permission involved	Permission Discarded
(*i*)	Clean_Required	103	27
(*ii*)	Malicious_Required	90	40
(*iii*)	Clean_Used	37	93
(*iv*)	Malicious_Used	31	99

that were 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 showed that only a small set of permissions had support that were greater than 0.1 (10%), so we followed the previous studies [19–21] to set 0.05 as an acceptable value for minimum support threshold for all four sub-datasets in *CPPM*. The minimum support difference threshold was set to be 0.15 (15%) and applied to filter out itemsets that were highly contrasted between clean and malicious applications.

4.2 Contrast Permission Patterns

Among the generated permission patterns, 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 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 were correlated and differed between clean and malicious applications. Based on the experimental results, we recorded 56 *required* permission patterns that were unique to the malware dataset, 31 *used* permission patterns that only appeared amongst malware, 17 *required* permission patterns and 9 *used* permission patterns that were present in both clean and malware dataset. These findings are presented as permission patterns (described in Table 5) which are listed in Tables 6-10, and summarized below.

Unique Required Permission (URP) Patterns. In Table 6 and 7, we presented the permission patterns that were frequently required by the applications in our dataset. It should be noted that these *required* permission patterns were unique to the malware dataset only; hence the support value for the clean applications was 0.

In Table 6, the top 15 permission combinations, where the first permission in the listed patterns belonged to the *normal* permissions category, are presented. 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

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

Table 5. Types of Permission Patterns

Permission Patterns	Description
Unique Required Permission (URP)	*Required* permission patterns present only in malware dataset
Unique Used Permission (UUP)	*Used* permission patterns present only in malware dataset
Common Required Permission (CRP)	*Required* permission patterns present in both clean and malware datasets
Common Used Permission (CUP)	*Used* permission patterns present in both clean and malware datasets

support values 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 6.

In Table 7, we listed the 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.

Unique Used Permission (UUP) Patterns. In Table 8, the combinations of the *used* permissions that are unique to the malware dataset only are reported. 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.

Table 6. 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	$URPSet_{15}$

Table 7. Unique *Required* Permission Sets in Malware Dataset (*Dangerous/Signature/SignatureOrSystem Permissions*)

Permission Set	Support		Permission Set ID
	Clean	*Malware*	
$pms0002$, $pms0005$, $pms0020$	0	0.2690	$URPSet_{16}$
$pms0002$, $pms0004$, $pms0020$	0	0.2576	$URPSet_{17}$
$pms0002$, $pms0005$, $pms0023$	0	0.2307	$URPSet_{18}$
$pms0002$, $pms0004$, $pms0023$	0	0.2234	$URPSet_{19}$
$pms0030$, $pms0036$	0	0.3228	$URPSet_{20}$
$pms0021$, $pms0036$	0	0.3163	$URPSet_{21}$
$pms0031$, $pms0036$	0	0.2690	$URPSet_{22}$
$pms0029$, $pms0036$	0	0.2674	$URPSet_{23}$
$pms0021$, $pms0028$	0	0.2519	$URPSet_{24}$
$pms0008$, $pms0030$	0	0.3269	$URPSet_{25}$
$pms0008$, $pms0021$	0	0.2894	$URPSet_{26}$
$pms0008$, $pms0031$	0	0.2649	$URPSet_{27}$
$pms0008$, $pms0029$	0	0.2429	$URPSet_{28}$
$pms0028$, $pms0036$	0	0.2413	$URPSet_{29}$
$pms0004$, $pms0006$, $pms0023$	0	0.4475	$URPSet_{30}$
$pms0004$, $pms0030$	0	0.3896	$URPSet_{31}$
$pms0004$, $pms0005$, $pms0020$	0	0.3106	$URPSet_{32}$
$pms0004$, $pms0021$	0	0.2462	$URPSet_{33}$
$pms0005$, $pms0013$	0	0.5453	$URPSet_{34}$
$pms0005$, $pms0031$	0	0.5094	$URPSet_{35}$
$pms0005$, $pms0021$	0	0.4190	$URPSet_{36}$

Another interesting observation is the presence of the `READ_LOGS` ($pms0011$) permission in over half of the permission patterns presented in Table 8. It is often combined with the `INTERNET` ($pms0001$) and `ACCESS_FINE_LOCATION` ($pms0002$) permissions. The remaining patterns include combinations of network-related and SMS-related permissions.

Common Required Permission (CRP) Patterns. Previously, we presented the permission patterns that were unique to malicious applications only. In Table 9, we listed the permission combinations that appeared in both clean and

Table 8. Unique *Used* Permission Sets in Malware Dataset

Permission Set	Support		Permission Set ID
	Clean	*Malware*	
$pms0001$, $pms0005$, $pms0006$, $pms0007$	0	0.5542	$UUPSet_1$
$pms0001$, $pms0005$, $pms0011$	0	0.4687	$UUPSet_2$
$pms0001$, $pms0006$, $pms0011$	0	0.4320	$UUPSet_3$
$pms0005$, $pms0006$, $pms0011$	0	0.4312	$UUPSet_4$
$pms0001$, $pms0007$, $pms0011$	0	0.4149	$UUPSet_5$
$pms0005$, $pms0007$, $pms0011$	0	0.4133	$UUPSet_6$
$pms0006$, $pms0007$, $pms0011$	0	0.3855	$UUPSet_7$
$pms0001$, $pms0002$, $pms0005$, $pms0007$	0	0.3423	$UUPSet_8$
$pms0001$, $pms0021$	0	0.3358	$UUPSet_9$
$pms0001$, $pms0002$, $pms0011$	0	0.2845	$UUPSet_{10}$
$pms0002$, $pms0005$, $pms0011$	0	0.2845	$UUPSet_{11}$
$pms0001$, $pms0002$, $pms0006$, $pms0007$	0	0.2829	$UUPSet_{12}$
$pms0002$, $pms0005$, $pms0006$, $pms0007$	0	0.2829	$UUPSet_{13}$
$pms0002$, $pms0006$, $pms0011$	0	0.2755	$UUPSet_{14}$
$pms0001$, $pms0020$	0	0.2600	$UUPSet_{15}$

Table 9. 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	$CRPSet_3$
$pms0001$, $pms0023$	0.0505	0.6349	−0.5844	$CRPSet_4$
$pms0023$	0.0522	0.6349	−0.5827	$CRPSet_5$
$pms0006$, $pms0023$	0.0399	0.6031	−0.5632	$CRPSet_6$
$pms0005$, $pms0006$	0.2421	0.7905	−0.5485	$CRPSet_7$
$pms0001$, $pms0005$, $pms0006$	0.2421	0.7897	−0.5477	$CRPSet_8$
$pms0001$, $pms0004$, $pms0005$	0.1328	0.6544	−0.5216	$CRPSet_9$
$pms0004$, $pms0005$	0.1337	0.6553	−0.5216	$CRPSet_{10}$
$pms0004$, $pms0005$, $pms0006$	0.1149	0.5623	−0.4474	$CRPSet_{11}$
$pms0004$, $pms0023$	0.0293	0.4637	−0.4344	$CRPSet_{12}$

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. We identified four permissions: `INTERNET` ($pms0001$), `READ_PHONE_STATE` ($pms0005$), `ACCESS_NETWORK_STATE` ($pms0006$) and `ACCESS_WIFI_STATE` ($pms0023$) that were present in different permission combinations and appeared in more than 40% of the malware dataset.

Common Used Permission (CUP) Patterns. In Table 10, we presented the *used* permission combinations that appeared in both the clean and malware datasets. Although both datasets had the same permission patterns, the ones in the malware dataset have higher support values. The patterns include the following permissions: `INTERNET` ($pms0001$), `READ_PHONE_STATE` ($pms0005$), `ACCESS_NETWORK_STATE` ($pms0006$), `VIBRATE` ($pms0007$) and lastly, `READ_LOGS` ($pms0011$). The same support difference for $CUPSet_1$ and $CUPSet_2$ indicated that the occurrence of these permission combinations are highly relevant. Moreover, we observed that even though `READ_LOGS` ($pms0011$) permission did not

Table 10. Common *Used* Permission Sets in Both Clean and Malware Datasets

Permission Set	Support		Difference	Permission Set ID
	Clean	*Malware*		
*pms*0001, *pms*0005	0.2991	0.9152	−0.6161	$CUPSet_1$
*pms*0005	0.3032	0.9169	−0.6137	$CUPSet_2$
*pms*0001, *pms*0005, *pms*0006	0.2363	0.7718	−0.5355	$CUPSet_3$
*pms*0005, *pms*0006	0.2363	0.7718	−0.5355	$CUPSet_4$
*pms*0001, *pms*0005, *pms*0007	0.2168	0.6512	−0.4344	$CUPSet_5$
*pms*0005, *pms*0007	0.2192	0.6528	−0.4336	$CUPSet_6$
*pms*0005, *pms*0011	0.0538	0.4686	−0.4148	$CUPSet_7$
*pms*0011	0.0693	0.4760	−0.4067	$CUPSet_8$
*pms*0001, *pms*0011	0.0685	0.4711	−0.4026	$CUPSet_9$

appear in the common *required* permission patterns, but it appeared in three common *used* permission patterns `READ_LOGS`, $CUPSet_7$ - $CUPSet_9$.

4.3 Discussion

Observations from Statistical Analysis. From our statistical analysis in Section 3.2, we observed that the `INTERNET` permission remained the most required (97.72%) and used (94.62%) permission in our experimental dataset. We also found, from Tables 1 and 2, that there was a significant difference in the frequencies of required and used permissions for the clean and the malware datasets. This further confirmed the observation made by Felt et al. in [22] that both clean and malicious applications can be over-privileged. Till date, most of the proposed solutions have only considered *required* permissions 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 malware detection.

Observations from Contrast Permission Patterns. In Section 4.2, we present the most significant permission sets generated by contrast mining. We found that a large number of *required* and *used* permission sets were unique in malicious applications only. The same permission sets were non-existent in clean applications, as shown by the 0 support value. This is a good indication that the contrast permission sets can be further applied during the malware detection phase to identify malicious applications. For *normal* required permissions, we observed from Table 6 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 deduced 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 [23], 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 7, we noticed several interesting permission sets on which we provide further explanation. For permission set IDs $URPSet_{16}$ and $URPSet_{17}$, we found that 25%

of malicious applications required both `ACCESS_FINE_LOCATION` (*pms*0002) and `ACCESS_COARSE_LOCATION` (*pms*0020) permissions. While *pms*0002 is used to access to GPS location sources, *pms*0020 is used for location information related to network sources. However, the documentation [24] 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 [25].

For the *used* permission sets that were unique in our malware dataset (Table 8), we observed that the permission set: `INTERNET` (*pms*0001), `READ_PHONE_STATE` (*pms*0005), `ACCESS_NETWORK_STATE` (*pms*0006), `VIBRATE` (*pms*0007) with permission set ID $UUPSet_1$ was used by 55% of the malware samples. Interestingly, the same permission set can be found in Table 6 under the permission set ID $URPSet_{11}$, with the exception that it was required by only 35% of the malware samples.

Moreover, it can be noted from Table 8 that the `READ_LOGS` (*pms*0011) permission was frequently associated with the permission sets and appeared in 25% to 50% of the malware dataset. There was previously no indication that the `READ_LOGS` (*pms*0011) permission 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 9 and 10. The negative support difference given in the table shows that the permission sets were more prevalent in malicious applications than in clean ones. We observed that the top two permission sets, $CRPSet_1$ and $CRPSet_2$ in Table 9 and $CUPSet_1$ and $CUPSet_{11}$ in Table 10 are the same.

5 Conclusion

Android uses a permission system to control access to restricted resources on smartphones. The permissions are indicative of the characteristics of an applications and as such, can be used to differentiate clean applications from malicious ones. However, most of the existing work only focused on *required* permissions and there is no extensive work on understanding key similarities and differences in permission patterns between clean and malicious applications.

To address these aforementioned issues, in this paper we combined both *required* and *used* permissions to identify a set of unique and common contrast permission patterns. Additionally, an efficient pattern mining method that can identify contrasting permission patterns for our clean and malware datasets was proposed. We observed that some permission sets were common in both datasets, while others were unique to only the clean or the malicious dataset.

By applying support value to the set of permission patterns, we filtered out the permission combinations that are less significant. Compared to Frank et

al.'s work [3] where the authors had to simulate permission request data to test their generated patterns, we applied our proposed methodology to combine the *required* and *used* permissions and retained those which can be used to contrast clean and malicious applications. 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. Orozco, A.: Is Google Acknowledging Android is not Secure? Malwarebytes (June 2013), http://blog.malwarebytes.org/intelligence/2013/06/is-google-acknowledging-android-is-not-secure-hmm/
2. Gilbert, P., Byung-Gon, C., Landon, P.C., Jaeyeon, J.: 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), Washington, USA, pp. 21–26 (June 2011)
3. Frank, M., Dong, B., Felt, A.P., Song, D.: Mining permission request patterns from Android and Facebook applications. In: Proceedings of the IEEE International Conference on Data Mining, Brussels, Belgium (ICDM 2012), pp. 1–16 (December 2012), http://arxiv.org/abs/1210.2429
4. Felt, A.P., Ha, E., Egelman, S., Haney, A., Chin, E., Wagner, D.: Android permissions: User attention, comprehension and behavior. In: Proceedings of the Symposium on Usable Privacy and Security (SOUPS 2012), Washington, D.C., vol. 3, pp. 1–14 (July 2012)
5. Zhou, Y., Jiang, X.: Dissecting Android malware: Characterization and evolution. In: Proceedings of the IEEE Symposium on Security and Privacy (SP 2012), San Francisco, CA, pp. 95–109 (May 2012)
6. Felt, A.P., Greenwood, K., Wagner, D.: The effectiveness of application permissions. In: Proceedings of the USENIX Conference on Web Application Development (WebApps 2011), Portland, Oregon, pp. 1–12 (June 2011)
7. Chia, P.H., Yamamoto, Y., Asokan, N.: 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), Lyon, France, pp. 311–320 (April 2012)
8. International Secure Systems Lab. Andrubis: Analyzing Android binaries, http://anubis.iseclab.org (accessed in May 2012)
9. Open Handset Alliance. Android, http://www.openhandsetalliance.com/android_overview.html (accessed in November 2007)
10. Ableson, F.: Introduction to Android development, http://www.ibm.com/developerworks/library/os-android-devel (accessed in May 2009)
11. Google. Google play, https://play.google.com (accessed in December 2012)
12. Google. Android permissions, http://developer.android.com/guide/topics/manifest/permission-element.html (accessed in December 2012)
13. Bartel, A., Klein, J., Monperrus, M., Traon, Y.L.: 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), Essen, Germany, pp. 274–277 (September 2012)

14. Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Bringas, P.G., Álvarez, G.: PUMA: Permission usage to detect malware in android. In: Herrero, Á., et al. (eds.) Int. Joint Conf. CISIS'12-ICEUTE'12-SOCO'12. AISC, vol. 189, pp. 289–298. Springer, Heidelberg (2013)
15. Sahs, J., Khan, L.: A machine learning approach to Android malware detection. In: Proceedings of the European Intelligence and Security Informatics Conference (EISIC 2012), Odense, Denmark, pp. 141–147 (August 2012)
16. Wu, D.J., Mao, C.H., Wei, T.E., Lee, H.M., Wu, K.P.: DroidMat: Android malware detection through manifest and API calls tracing. In: Proceedings of the 2012 Seventh Asia Joint Conference on InformationSecurity (Asia JCIS 2012), Tokyo, Japan, pp. 62–69 (August 2012)
17. Zhou, Y., Wang, Z., Zhou, W., Jiang, X.: Hey, You, Get Off of My Market: Detecting Malicious Appsin Official and Alternative Android Markets. In: Proceedings of the 19th Annual Network and Distributed System Security Symposium (NDSS 2012), San Diego, California, pp. 1–13 (February 2012)
18. Agrawal, R., Imieinski, T., Swami, A.: Mining association rules between sets of items in large databases. In: Buneman, P., Jajodia, S. (eds.) Proceedings of the ACM SIGMOD International Conference on the Managementof Data, Washington, D.C., pp. 207–216. ACM Press (1993)
19. Liu, S., Law, R., Rong, J., Li, G., Hall, J.: Analyzing changes in hotel customers' expectations by trip mode. International Journal of Hospitality Management (2012) (in press)
20. Rong, J., Vu, H.Q., Law, R., Li, G.: A behavioral analysis of web sharers and browsers inhong kong using targeted association rule mining. Tourism Management 33(4), 731–740 (2012), http://dx.doi.org/10.1016/j.tourman.2011.08.006
21. Law, R., Rong, R., Vu, H.Q., Li, G., Lee, H.A.: Identifying changes and trends in hong kong outbound tourism. Tourism Management 32(5), 1106–1114 (2011)
22. Felt, A.P., Chin, E., Hanna, S., Song, D., Wagner, D.: Android permissions demystified. In: Proceedings of the ACM Conference and Communications Security (CCS 2011), Chicago, USA, pp. 627–638 (October 2011)
23. F-Secure. Trojan:android/droidkungfu.c, http://www.f-secure.com/v-descs/trojan_android_droidkungfu_c.shtml (accessed in January 2013)
24. Android Developer. Location strategies, http://developer.android.com/guide/topics/location/strategies.html (accessed in January 2013)
25. Chin, E., Felt, A.P., Greenwood, K., Wagner, D.: Analyzing inter-application communication in Android. In: Proceedings of the 9th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys 2011), Washington, USA, pp. 239–252 (June 2011)