Permission-Combination-based Scheme for Android Mobile Malware Detection

Shuang Liang and Xiaojiang Du
Dept. of Computer and Information Science
Temple University,
Philadelphia, PA 19121, USA
{shuang.liang2012, dux}@temple.edu

***Abstract*—With the increase use of Android mobile phones, more Android malwares are being developed. Android malware detection becomes a crucial task. In this paper, we present a permission-combination-based scheme for Android malware detection. The Android malware detection scheme is based on permission combinations declared in the application manifest file. We obtain the permission combinations that are requested frequently by malwares but rarely by benign applications. We generate rule sets based on the permission combinations. Our experimental results show that the malware detection rate is up to 96%, and the benign application recognition rate is up to 88%. Our experimental results with real malwares show that the Android malware detection scheme is very efficient and effective.**

***Keywords*—Android; mobile phone; malware detection**

I. Introduction

Android is the world's most popular mobile platform released by Google [1]. Unfortunately, the popularity of this platform also attracts malware developers. Lots of notorious malwares have been seen in the wild such as Geinimi [2], DroidKungFu [3] and AnserverBot [4]. The malwares spread quickly in the Android market and cause financial loss or privacy leaking for mobile users. The openness of Android market allows malwares being published on the third-party Android markets without much inspection. This promotes the growth of Android malware.

Android has an extensive API and permission enforcement system. Permission checks for Android system APIs, Content Providers, and Intents are enforced by Android system [5]. Application developers must declare the permissions required by the system APIs, Content Providers and Intents in their programs in order to use them. Android also has intall-time permission system. When installing applications, the user is notified about the permissions the applications require. Android users must grant the permissions requested by the application in order to run the program. Many literatures have discussed Android permission system [5], [6], [7], but none of them focuses on how to use the permissions to detect malware.

In this paper, we propose an effective security scheme - Droid Detective to detect Android malware based on permission combinations. Droid Detective can obtain permissioon combinations that are requested frequently by malwares but rarely by benign applications. Droid Detective can automatically generate rule sets for identifying malware. We use 1260 malware samples (of 49 malware families) published by NCSU researchers [8] and 741 benign applications (of 34 application categories) collected from the official Google Android Market [9] as our training and evaluation data set. The malware detection rate of Droid Detective is as high as 96%, and benign application recognition rate is as high as 88%. The details of Droid Detective is given in Section II.

The contributions of this paper are summarized as follows:

- Based on Android permission combinations, we design and implement an automatical-rule-generation scheme and an effective rule-based malware detection scheme - Droid Detective.
- We show the relevance between the generated rules and the corresponding malware behaviors.
- We implement Droid Detective and conduct experiments with real malware samples, which shows our scheme is very efficient and effective.

The rest of this paper is organized as follows: In Section II, we introduce Droid Detective - a permission-combination-based scheme. we present the permission combination mapping process and the rule sets selection process in Section III. In Section IV, we give the experimental evaluation of the permission-combination-based scheme. We summarize the lessons learned from Droid Detective in Section V. We discuss the related works in Section VI and conclude this paper in Section VII.

II. Overview of Permission-combination-based Scheme - Droid Detective

Android permission check system requires application developers declaring permissions used in their applications (Apps) in *AndroidManifest.xml* file, in order to invoke Android API successfully. The declared permissions are useful and effective to reveal the potential risks of Apps being installed [10]. Permission-based malware detection techniques are widely used in the literatures. Jiang and Zhou from NCSU published their analysis results based on 1,260 Android malware samples collected from a variety of Android Markets over more than one year [11]. They compared the top 20 permissions requested by malware samples and those requested by benign Apps on Google Android Market [9]. The results show that permissions such as INTERNET, READ_PHONE_STATE, ACCESS_NETWORK_STATE and WRITE_EXTERNAL_STORAGE are requested frequently in both malware and benign Apps. On the other hand, permissions such as READ_SMS, WRITE_SMS, SEND_SMS and RECEIVE_SMS are overwhelmingly requested by malwares but rarely by benign Apps. Zhou et al. implemented permission-based behavioral footprinting scheme in DroidRanger [12].

978-1-4799-2003-7/14/$31.00 ©2014 IEEE

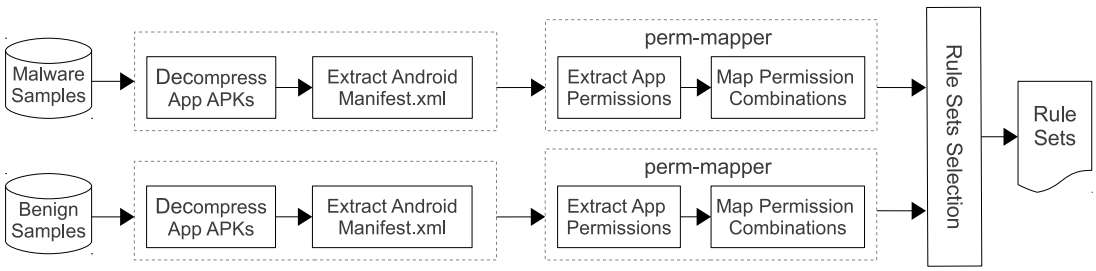


Fig. 1. Permission Combination Rule Generation Process

DroidRanger uses the permission features of malware to filter some potential malwares. Inspired by the above work, we develop a tool called k-map, which is also based on permission features declared in Android manifest files. Different from DroidRanger and others, k-map inspects the combinations of k (k≥1) permissions instead of a single permission as in DroidRanger. What's more, instead of manually inspecting permission features of certain malware family, k-map automatically generates rule sets based on permission combinations. K-map iteratively reads application package files as input and calculates the permission request frequency in the form of k (k = 1, 2, 3, ...) permission combinations.

Figure 1 shows the rule set generation process of k-map. The samples are in the compressed format of Android application package (APK). First, the samples are decompressed into several files. The decompressed files include dalvik executable file called *class.dex*, a compressed application manifest file *AndroidManifest.xml* and other resource and library files. After decompressing the APKs, k-map converts the compressed manifest files to readable XML format. The converted manifest XML files are used as input to perm-mapper, which is used to calculate and map permission combinations with their request frequencies. Perm-mapper extracts permissions from the XML files. Then it creates and updates the values corresponding the permission combinations in the map. The values are the accumulated request times that each permission combination has been requested. Finally, request frequencies of all permission combinations are calculated. These procedures are applied to both malwares and benign Apps and iterated over k ranging from 1 to 6 in our experiments. The frequencies of permission combinations are further fed into the selection process. The selection process can find out the permission combinations that are highly requested by malwares but rarely by benign Apps. The selected permission combinations are used as rule sets. We use these rule sets to detect potential malwares. Section III discusses the details about the permission-combination-based malware detection scheme and the notations used in algorithms and equations are listed in Table I.

III. Permission Combination Mapping and Rule Sets Selection

For malwares without root exploits, permissions must be declared in per application manifest file before being used. Most of the malwares require some permission combinations in order to carry out attacks. Based on the above observation, we develop a tool called k-map to find permission combinations that are frequently used by malwares.

TABLE I. Notations Definitions

Notation	Description
k-map	Permission combination and request freq. map structure
MapList	List of all k-maps for all the malware families
sampleCnt	Number of samples that have been processed
MIN_{diff}	Minimum freq. difference between malware and benign Apps.
R	Generated rule sets
MIN_s	Minimum number of samples eligible for k-map
F_m	Permission combination request freq. in malware dataset
F_b	Permission combination request freq. in benign dataset
R_m	# of malwares that have the permission request combination
R_b	# of benign Apps that have the permission request combination
N_m	Total number of malware samples
N_b	Total number of benign Apps

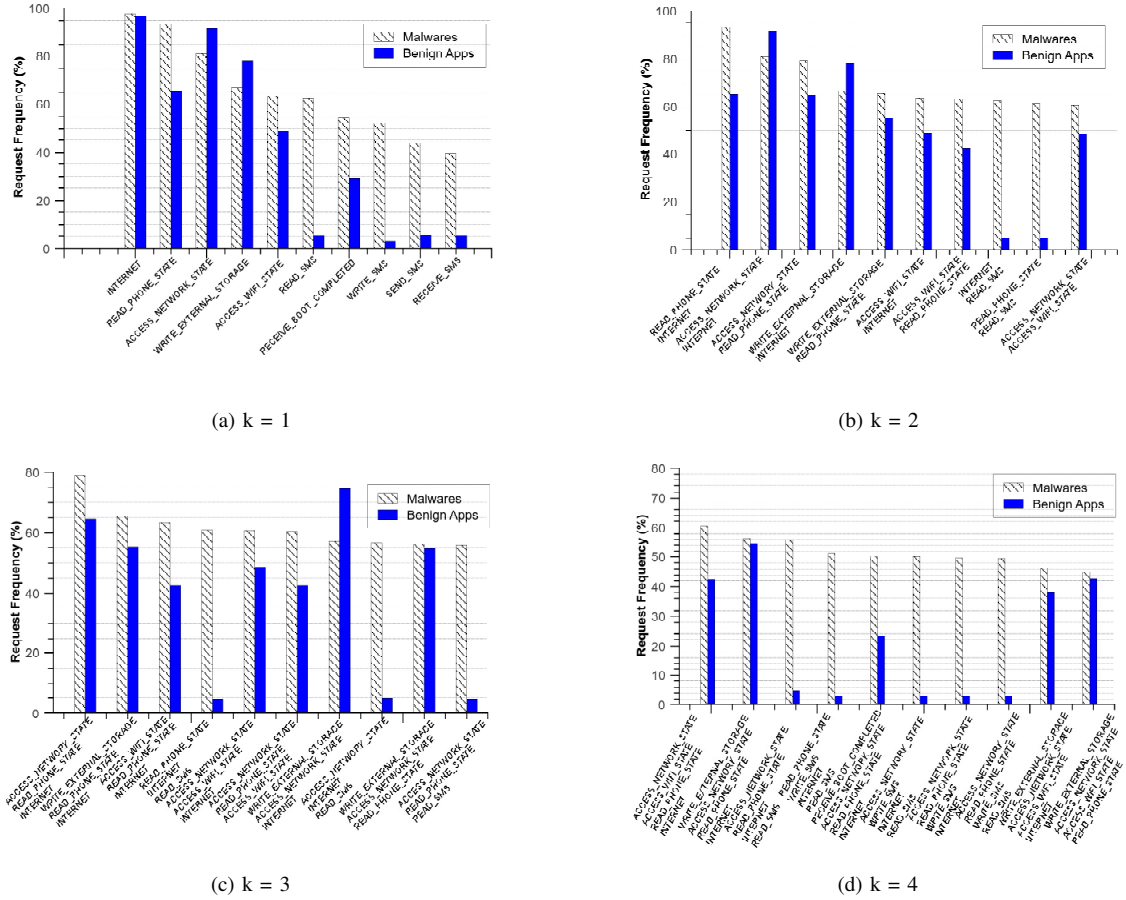
Algorithm 1 Generating Permission Map When k = 2

1: $k \leftarrow 2$; $k\text{-}map \leftarrow \{\}$
2: $CombList \leftarrow$ All combinations with k permissions
3: **for all** $comb \in CombList$ **do**
4: **if** $comb$ is in $k\text{-}map$ **then**
5: $k\text{-}map(comb) = k\text{-}map(comb) + 1$
6: **else**
7: $k\text{-}map = k\text{-}map \cup <comb, 1>$
8: **end if**
9: **end for**

K-map requires two kinds of datasets to generate the permission maps, i.e., malware samples and benign application samples. For the malware part, we use the Android malware dataset released by NCSU researchers through the Android Malware Gnome Project [8]. This dataset for Android malware includes most malwares that appear in the market. It has 1260 malware samples in 49 different malware families dated from August 2010 to October 2011. For the benign application dataset, we build a tool to crawl free Apps from the Google Android Market [9]. We collect 741 Apps randomly in 34 different categories from the Google Android Market.

The mapping process includes reading all the *AndroidManifest.xml* files, extracting permission combinations and calculating the permission request frequencies out of all permission combinations. Algorithm 1 shows the steps for generating a permission map when k = 2. In the algorithm, we read all the combinations with k permissions in the Android manifest file into a list. Afterwards, we update the counter in the map for each combination in the list. The algorithms used to create 3-map, 4-map ... are similar.

To generate k-map for malwares and benign Apps, we apply Alg. 1 iteratively for each manifest file of malwares and benign Apps. We generate k-maps for k from 1 to 6. The top 10 permission combinations for malwares when k = 1,



(a) k = 1

(b) k = 2

(c) k = 3

(d) k = 4

Fig. 2. Top 10 Requested Permission Combinations

2, 3, and 4 are shown in Fig. 2a, Fig. 2b, Fig. 2c and Fig. 2d, respectively. In those figures, the request frequencies of benign Apps are also drawn for comparison. Due to the limited space, we only list the top 5 permission combinations for k = 5 and k = 6 in Table II and Table III, respectively. When k = 1, malwares request permissions READ_SMS, WRITE_SMS, SEND_SMS and RECEIVE_SMS much more frequently than benign Apps. These are relatively normal permissions. When k = 2, we find two permission combinations that are requested apparently more frequently by malwares than by benign Apps: INTERNET & READ_SMS, READ_PHONE_STATE & READ_SMS. With the INTERNET and READ_SMS permission combination, malwares can send out private SMS messages by the Internet; With READ_PHONE_STATE and READ_SMS permission combination, malwares are able to read phone IDs such as IMEI (International Mobile Equipment Identity) and IMSI (International Mobile Subscriber Identity), as well as SMS messages, which can be used to identify mobile users and gather personal information. With a larger k, the obtained permission combinations are more invasive. We find that: the larger the value of k, the more accurate the permission combinations are able to identify malwares. Notice that when k = 1, the results are consistent with the top 20 permissions requested by 1260 malwares and benign Apps published by NCSU researchers [11]. We investigate the results for k = 2, 3, 4, 5 and 6, and find that some permission combinations are frequently requested by both malicious and benign Apps. While some permission combinations are apparently requested more often by malwares than by benign Apps. Almost all the permission combinations that are frequently requested by malwares include SMS related permissions, such as WRITE_SMS and READ_SMS. We suspect the reason is that most samples in the malware dataset are related to SMS attacks, such as intercepting and leaking SMS messages, sending SMS messages to premium numbers. After checking the dataset information published by NCSU researchers, our guessing is confirmed. According to their results [11], there are 571 malware samples signing up premium service by sending SMS messages and 315 samples blocking incoming SMS confirmation messages. In addition to those samples, there are 138 malware samples stealing personal information via SMS messages. In order to find out permission combinations that are specific to the the non-SMS malware families, we modify k-map and let k-map apply mapping and selection processes to each malware family instead of to the entire dataset.

Different malware families usually need different permission combinations to carry out their attacks. In order to identify as more malware families as possible, we revise k-map to support creating rule sets for each malware family. Then we add up all the sets of permission combinations for each malware family and use the final result as the rule sets of all the malware families in the dataset. The number of samples in each malware family ranges from 1 to 309. It is hard to

Algorithm 2 Generating k-map for All Malware Families

```
1: MIN_s ← 10
2: MapList ← {}
3: for all family ∈ malware dataset do
4:    new k-map
5:    sampleCnt ← 0
6:    for all sample ∈ family do
7:       decompress sample APK file
8:       inflate AndroidManifest.xml to plain xml file
9:       obtain all k combinations from xml file
10:      update k-map
11:      sampleCnt = sampleCnt + 1
12:   end for
13:   if sampleCnt ≥ MIN_s then
14:      MapList = MapList ∪ k-map
15:   end if
16: end for
```

generate a working k-map for malware families with very few samples (like 1 sample). Therefore, we set the minimum number of samples $MIN_s = 10$ as the eligible malware families for k-map. Malware families with less than MIN_s malware samples are not considered. Algorithm 2 illustrates the steps of the revised per family k-map. This algorithm creates k-map for each malware family. Instead of reading and updating permission combinations of the malware samples altogether, the algorithm processes one malware family a time and generates the corresponding permission k-map for each malware family. For the "update k-map" operation in the algorithm refer to Alg. 1. K-maps for all the malware families are used to generate rule sets covering all the eligible malware families appeared in the malware dataset.

The rule sets selection process is to extract the permission combinations that are requested more often by malwares than by benign Apps. K-map rule generator reads permission maps of all the malware families and the map of benign Apps. First, k-map calculates the permission request frequency F_m for all the permission combinations in each malware family. Second, k-map calculates the permission request frequency F_b of corresponding benign Apps. Third, k-map obtains the frequency difference $D = F_m - F_b$. Equation 1 shows how to calculate the frequency difference for a permission combination. The notations in Equation 1 are described in Table I. We use the difference value (MIN_{diff}) as a control parameter in this scheme. By conducting experiments with various values of MIN_{diff}, we find that when $MIN_{diff} = 90\%$, the selected permission combinations are enough to detect most malwares while achieving high identification rate for benign Apps. So we set the minimum difference $MIN_{diff} = 90\%$ to select permission combinations highly requested in malwares but rarely requested by benign Apps. Algorithm 3 shows how to generate the permission-based rule sets.

$$\begin{aligned} F_m &= \frac{R_m}{N_m} \\ F_b &= \frac{R_b}{N_b} \\ D &= F_m - F_b \end{aligned} \tag{1}$$

TABLE II. TOP 5 PERMISSION COMBINATIONS WHEN K = 5

Permisson Combination	Malware (%)	Benign (%)
ACCESS_NETWORK_STATE INTERNET READ_PHONE_STATE READ_SMS WRITE_SMS	49.52	3.1
ACCESS_NETWORK_STATE ACCESS_WIFI_STATE INTERNET READ_PHONE_STATE WRITE_EXTERNAL_STORAGE	44.68	37.92
ACCESS_NETWORK_STATE ACCESS_WIFI_STATE INTERNET READ_PHONE_STATE RECEIVE_BOOT_COMPLETED	41.03	19.16
ACCESS_NETWORK_STATE ACCESS_WIFI_STATE INTERNET READ_PHONE_STATE READ_SMS	40.87	3.78
ACCESS_NETWORK_STATE ACCESS_WIFI_STATE INTERNET READ_SMS WRITE_SMS	38.57	2.56

TABLE III. TOP 5 PERMISSION COMBINATIONS WHEN K = 6

Permisson Combination	Malware (%)	Benign (%)
ACCESS_NETWORK_STATE ACCESS_WIFI_STATE INTERNET READ_PHONE_STATE READ_SMS WRITE_SMS	38.25	2.56
ACCESS_NETWORK_STATE INTERNET READ_PHONE_STATE READ_SMS WRITE_EXTERNAL_STORAGE WRITE_SMS	29.84	3.1
ACCESS_NETWORK_STATE INTERNET READ_PHONE_STATE READ_SMS VIBRATE WRITE_SMS	29.76	3.1
ACCESS_NETWORK_STATE INTERNET READ_PHONE_STATE READ_SMS SEND_SMS WRITE_SMS	28.97	2.56
ACCESS_NETWORK_STATE INTERNET READ_PHONE_STATE READ_SMS RECEIVE_SMS SEND_SMS	28.73	3.1

For each malware family whose number of samples are greater than or equal to MIN_s, we generate a rule referred to as $R_n(n = 1, 2, 3 \ldots)$, which includes the permission combinations whose request frequency by malwares is MIN_{diff} more than that of benign Apps. The final rule sets for detecting malware is the union of the rules of all the malware families, namely, $R = R_1 \cup R_2 \cup R_3 \ldots \cup R_n$. Figure 3 lists part of the selected rule sets for malware family Geinimi when k = 3. In the generated rule sets for Geinimi, some permission combinations show the apparent malicious intensions of the malware family. For example, rule set "ACCESS_FINE_LOCATION INTERNET WRITE_CONTACTS" can be used by malware to send location information to an adversary via Internet.

Algorithm 3 Generate Rule Sets

$MIN_{diff} \leftarrow 90\%$
$R \leftarrow \{\}$
for all k-map $\in$ malware k-map list **do**
  for all comb $\in$ top 20 k-map permission combinations **do**
    $freq1 \leftarrow R_m(comb)/sampleNumInFamily$
    $freq2 \leftarrow R_b(comb)/sampleNumInBenign$
    $diff = freq1 - freq2$
    if $diff \geq MIN_{diff}$ **then**
      $R = R \cup comb$
    end if
  end for
end for

```
@Geinimi
MOUNT_UNMOUNT_FILESYSTEMS READ_PHONE_STATE SET_WALLPAPER
INTERNET SET_WALLPAPER WRITE_HISTORY_BOOKMARKS
READ_CONTACTS READ_HISTORY_BOOKMARKS READ_PHONE_STATE
INTERNET SEND_SMS WRITE_HISTORY_BOOKMARKS
ACCESS_FINE_LOCATION INTERNET WRITE_CONTACTS
ACCESS_FINE_LOCATION INSTALL_SHORTCUT INTERNET
INTERNET READ_HISTORY_BOOKMARKS WRITE_EXTERNAL_STORAGE
READ_CONTACTS SET_WALLPAPER WRITE_CONTACTS
READ_PHONE_STATE SEND_SMS WRITE_HISTORY_BOOKMARKS
```

Fig. 3. Part of the Rule Sets for Malware Family Geinimi

TABLE IV. DATASET INFORMATION

Dataset	# of Families	Sample #	k-map	Evaluation
Malware	49	1260	190	980
Benign Apps	34	741	340	401

IV. EXPERIMENTAL EVALUATION AND DISCUSSION

Both the malware dataset and benign dataset are divided into two groups: one for k-map samples and the other for evaluation (testing). Table IV shows the statistics of the dataset. Column 2 is the number of families of the Malware (Benign) Apps. Column 3 is the total number of samples. Column 4 is the number of samples used to generate the k-map rule sets, and column 5 is the number of samples used for evaluation. We define the percentage of malware samples detected by the rules as the hit rate (HR) and the percentage of Apps in the benign samples that pass the rule check as the pass rate (PR). The false negative rate (FNR) is the percentage of malwares that are not detected, and the false positive rate (FPR) is the percentage of benign Apps that are considered as malware. Table V shows the results of the experimental evaluation.

As we can see from Table V, in general, when k increases, the hit rate (HR) decreases (because more permissions need to be satisfied in order to be declared as a malware), but the pass rate (PR) of benign Apps increases. There is a tradeoff between the HR of malware and the PR of benign Apps. When k = 6, the HR is 83.57% and the false positive rate (FPR) is $1 - 87.53\% = 12.47\%$, which means that k-map can detect most (83.57%) of the malwares, and it has a low false positive rate of 12.47%. That is, when k = 6, k-map achieves pretty good performance in malware detection.

After investigating the rule sets generated by k-map, we find that most of the malwares that can cause financial loss are premium-rate SMS related. The results are consistent with the data published by NCSU [11]. In their 1260 collected malwares, there are 571 malware samples signing up premium service by sending SMS messages, 315 samples blocking incoming SMS confirmation messages and 138 malware samples stealing personal information via SMS messages. In the real world, SMS related malwares and trojan applications are spreading and upgraded rapidly [13], [14], [15], [16], [17], [18]. This kind of malware can send premium SMS messages in the background and intercept incoming SMS messages from service providers without being noticed by the user. Our permission-combination-based scheme can detect the SMS related malwares efficiently with low false positive rate.

TABLE V. EVALUATION RESULTS

k	HR (%)	PR (%)	FNR (1-HR) (%)	FPR (1-PR) (%)
1	96.63	49.38	3.37	50.62
2	95.82	44.11	4.18	55.89
3	94.39	55.36	5.61	44.64
4	94.49	61.35	5.51	38.65
5	90.51	61.10	9.49	38.90
6	83.57	87.53	16.43	12.47

V. LESSONS LEARNED FROM DROID DETECTIVE

Some of the lessons we learned from Droid Detective are summarized as follows:

- Be aware of some permission combinations, e.g., App requesting ACCESS_COARSE_LOCATION and INTERNET permissions can expose your locations to others via the Internet connection; and App requesting INTERNET and READ_CONTACTS permissions can leak your contacts information via the Internet connection.
- Be careful when installing Apps from alternative Android Apps Market. Some popupar Apps can be repackaged with trojans. We suggest downloading and installing Apps only from the authorized Android Markets.
- Be cautious when you install Apps that request WRITE_SMS and SEND_SMS permissions. They may send SMS messages to premium-rate numbers and cause you pay high bills.
- Check your mobile device bills regularly and look for the services you do not know.

VI. RELATED WORK

There are two approaches to analyze and detect malware: static analysis and dynamic analysis. Static analysis is used to find some suspicious patterns of certain malware family. This method works on the application source code or binaries. Dynamic analysis is based on the behavior features of the applications which requires running the applications in a controlled or isolated environment. Zhou et al. proposed a permission-based behavioral footprinting scheme to detect new samples of known Android malware families [12]. They use permissions to filter out some irrelevant Apps that do not request specific permissions. In their approach, permission is used to reduce the number of applications for further complex analysis. Then the complex analysis only needs to handle

applications that pass the first step. They also use behavioral footprint to match some behavioral features of malware, such as the registered broadcast listener in the application. Our schemes are different from them in that we focus on finding out permission combinations that requested frequently by malwares but rarely by benign Apps, and we use these combinations as features to detect malware.

Enck, Ongtang et al. presented Kirin [19]. They modified Android Application Installer, and extended it with security certification at install time. They use permission combinations as part of their security rules. Kirin relies on well constructed security rules to be effective. Defining security rules for Kirin requires a thorough understanding of threats and existing protection mechanisms which is usually done by security expert. Our research focus on how to select permission combinations that can identify malwares. Our research is orthogonal to their work.

Besides permission and signature based static analysis techniques, there are studies enhancing Android system security by information flow or data flow tracking such as TaintDroid [20] and D2Taint [21]. They use tag chunk to keep track of data in order to find information leakage at runtime. Information flow tracking needs lots of memory. None of these schemes is energy-efficient, hence they are not suitable for resource-constrained mobile platforms.

CrowdDroid [22] presented a framework that keeps monitoring Linux system calls as features and uses k-means clustering algorithm to differentiate benign applications from malicious trojan applications. The drawback of this method is also the high energy consumption. Monitoring system calls consume lots of resources of a mobile device. Isohara, Takemori et al. also proposed a kernel log analysis framework to detect malicious activity [23]. They use behavioral signatures to match log information, but there are only a limited number of signatures.

VII. Conclusion

In this paper, we studied the urgent issue of malware detection in Android system. We designed and implemented Droid Detective to enhance the security of Android system. Droid Detective is an offline tool that utilizes permission combinations to detect malwares. We generated permission combinations for k = 1, 2, 3, 4, 5, and 6, and showed the relevance between permission combinations and malicious behaviors of malwares. Our experiments with real malware samples showed that Droid Detective can effectively detect malware with low false positive and false negative rates.

Acknowledgment

This research was supported in part by the US National Science Foundation (NSF) under grants CNS-0963578, CNS-1022552, and CNS-1065444.

References

[1] "Android," http://www.android.com/.

[2] "Geinimi," http://www.symantec.com/security_response/writeup.jsp?docid=2011-010111-5403-99.

[3] X. Jiang, "Droidkungfu," http://www.csc.ncsu.edu/faculty/jiang/DroidKungFu.html.

[4] X. Jiang, "Anserverbot," http://www.csc.ncsu.edu/faculty/jiang/AnserverBot/.

[5] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, pp. 627–638.

[6] M. Nauman, S. Khan, and X. Zhang, "Apex: extending android permission model and enforcement with user-defined runtime constraints," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, 2010, pp. 328–332.

[7] R. Johnson, Z. Wang, C. Gagnon, and A. Stavrou, "Analysis of android applications' permissions," in *Proceedings of the 2012 IEEE Sixth International Conference on Software Security and Reliability Companion*, 2012, pp. 45–46.

[8] Y. Zhou and X. Jiang, "Android malware genome project," http://www.malgenomeproject.org/.

[9] "Google android market," https://play.google.com/store/apps?feature=corpus_selector.

[10] 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 Eighth Symposium on Usable Privacy and Security*, 2012, pp. 3:1–3:14.

[11] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy (SP)*, may 2012, pp. 95 –109.

[12] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets," in *Proceedings of the 19th Annual Network and Distributed System Security Symposium*, 2012.

[13] X. Jiang, "Security alert: New android malware – hipposms," http://www.csc.ncsu.edu/faculty/jiang/HippoSMS/.

[14] I. Asrar, "The day after the year in mobile malware?" http://www.symantec.com/connect/blogs/day-after-year-mobile-malware.

[15] "Detailed analysis of android.arspam," http://forensics.spreitzenbarth.de/2011/12/22/detailed-analysis-of-android-arspam/.

[16] X. Jiang, "Security alert: New beanbot sms trojan discovered," http://www.csc.ncsu.edu/faculty/jiang/BeanBot/.

[17] "Boxer sms trojan: Malware as a global service," http://www.welivesecurity.com/2012/11/29/android-boxer-a-worldwide-sms-trojan/.

[18] Symantec, "Android.dogowar," http://www.symantec.com/security_response/writeup.jsp?docid=2011-081510-4323-99.

[19] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in *Proceedings of the 16th ACM conference on Computer and communications security*, 2009, pp. 235–245.

[20] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, 2010, pp. 1–6.

[21] B. Gu, X. Li, G. Li, A. C. Champion, Z. Chen, F. Qin, and D. Xuan, "D2taint: Differentiated and dynamic information flow tracking on smartphones for numerous data sources," Technical Report, Tech. Rep., 2012.

[22] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for android," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, 2011, pp. 15–26.

[23] T. Isohara, K. Takemori, and A. Kubota, "Kernel-based behavior analysis for android malware detection," in *Proceedings of the 2011 Seventh International Conference on Computational Intelligence and Security*, 2011, pp. 1011–1015.