University of Nebraska - Lincoln
DigitalCommons@University of Nebraska - Lincoln

Computer Science and Engineering: Theses, Dissertations, and Student Research

Computer Science and Engineering, Department of

7-2016

Significant Permission Identification for Android Malware Detection

Lichao Sun
University of Nebraska-Lincoln, james.sun137@gmail.com

SIGNIFICANT PERMISSION IDENTIFICATION FOR ANDROID MALWARE DETECTION

by

Lichao Sun

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professors Witawas Srisa-an and Qiben Yan

Lincoln, Nebraska

August, 2016

SIGNIFICANT PERMISSION IDENTIFICATION FOR ANDROID MALWARE DETECTION

Lichao Sun, M.S.

University of Nebraska, 2016

Adviser: Witawas Srisa-an, Qiben Yan

A recent report indicates that a newly developed malicious app for Android is introduced every 11 seconds. To combat this alarming rate of malware creation, we need a scalable malware detection approach that is effective and efficient. In this thesis, we introduce SigPID, a malware detection system based on permission analysis to cope with the rapid increase in the number of Android malware. Instead of analyzing all 135 Android permissions, our approach applies 3-level pruning by mining the permission data to identify only significant permissions that can be effective in distinguishing benign and malicious apps. Based on the identified significant permissions, SigPID utilizes classification algorithms to classify different families of malware and benign apps. Our evaluation finds that only 25% of permissions (34 out of 135 permissions) are significant. We then compare the performance of our approach, using only 25% of all permissions, against a baseline approach that analyzes all permissions. The results indicate that when Support Vector Machine (SVM) is used as the classifier, we can achieve over 90% of precision, recall, accuracy, and F-measure, which are about the same as those produced by the baseline approach. We also show that SigPID is effective when used with 67 other commonly used supervised learning approaches. We find that 55 out of 67 algorithms can achieve F-measure of at least 85%, while the average running time can be reduced by 85.6% compared with the baseline approach. When we compare the detection effectiveness of SigPID to those of other approaches,

SigPID can detect 96.54% of malware in the data set while other approaches detect 3.99% to 96.41%.

ACKNOWLEDGMENTS

I want to give my thanks to my advisers Dr. Witawas Srisa-an and Dr. Qiben Yan. When I was an undergraduate student at UNL, I have no ideas about doing research in Computer Science. Dr. Witty discussed with me and gave me good advice to start my research at UNL. His kindness and guidance supports my three-year research study here. In Dr. Witty's research group, not only I learn tremendous amount of knowledge about research in security, I also learn other life-long skills such as profound thinking and interpersonal interactions. I really appreciate his continuous help and patience during my study, which gives me courage and confidence to conduct research in areas that I am interested in. At the same time, I also want to give my thanks to Dr. Qiben Yan. Without his guidance, I may not finish my first paper and my thesis. I have made a lot of improvement with my writing under his guidance. Under their supervision, I have a very enjoyable research experience at UNL.

I also want to give my thanks to my parents, who love me, support my choice, and teach me how to be a good person. Without them, I would not be half the man I am now. They teach me to find the happiness in my dreams, which is the most important reason I choose to start my graduate study at UNL and continue this journey towards my Ph.D.

Last but not the least, I want to give my thanks to my committee members who spending time, reading this thesis, attending my defense, and providing productive feedback. I want to give my thanks to the people give me help during my studies at UNL, such as my courses professors, my classmates and my friends in Lincoln. I also want to thank CSE, which provides my GTA support and provide their students with developing professional skills and setting career paths.

I love my life here, and I will remember these wonderful years forever.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Android is currently the most used smart-mobile device platform in the world, occupying 82.8% of market share [1]. As of now, there are nearly 2 million apps available for downloading from Google Play, and more than 50 billion downloads to date. Unfortunately, the popularity of Android also creates interests from cyber-criminals who create malicious apps that can steal sensitive information and compromise systems. Unlike other competing smart-mobile device platforms, such as iOS, Android allows users to install applications from unverified sources such as third party stores. This problem has been so serious that a recent report indicates that 97% of all mobile malware target Android devices [2]. In 2015 alone, over 2.3 million new malicious apps have been uncovered. This roughly translates to an introduction of a new malicious Android app every 11 seconds [3]. These malicious apps are created to perform different types of attacks in the form of trojans, worms, exploits, and viruses. Each type of malicious apps has at least 50 variants, which makes it extremely challenging to detect them all [4].

To address security concerns, researchers have used various approaches including static and dynamic analyses, machine learning, and data mining to develop malware detection tools. For example, RiskRanker [5] uses static analysis to discover malicious behaviors in Android apps. Stowaway [6] is another static analysis tool that detects overprivilege in Android apps. While these approaches can identify potential malicious code sections, they also tend to produce a large number of false positives. To increase the analysis precision, researchers have also used dynamic analysis to capture execution context. For example, TaintDroid [7] dynamically tracks multiple sensitive data source simultaneously. The weakness of TaintDroid is that manual efforts to traverse user interfaces are needed to cover dangerous functionality effectively. Dynamic analysis approaches, in general, also need effective input suites to adequately exercise execution paths.

As such, work by Petra's et al. [8] shows that dynamic program analysis alone is not enough to assure Android security. They present a broad range of anti-analysis techniques for malware to successfully evade dynamic analysis. Therefore, more efforts have recently been spent on using machine learning and data mining techniques to detect Android malware based on permissions. For example, DREBIN [9] combines static analysis and machine learning techniques to detect Android malware. The experimental result shows that DREBIN can achieve high detection accuracy by using as many features as possible to aid detection. However, using more features also increases the overhead of their system.

In this thesis, we present SigPID, an approach that extracts significant permissions from apps, and uses the extracted information to effectively detect malware using supervised learning algorithms. The design objective of SigPID is to detect malware *efficiently* and *accurately*. As stated earlier, the number of newly introduced malware is growing at an alarming rate. As such, being able to detect malware efficiently would allow analysts to be more productive in identifying and analyzing them. Our approach analyzes permissions and then identifies only the ones that are significant in distinguishing between malicious and benign apps. Specifically, we propose a multi-level data pruning approach including *permission ranking with negative rate*, *permission mining with association rules* and *support based permission ranking* to extract significant permissions strategically. Then, classification algorithms are used to classify different types of malware and benign apps.

The results of our empirical evaluation show that SigPID can cut down the number of permissions that we need to analyze by 75%, while maintaining over 90% malware detection accuracy and F-measure when Support Vector Machine (SVM) is used as the classifier. To show the generality of this approach, we also test SigPID with 67 commonly used supervised algorithms and find that it maintains very high accuracy with these algorithms. We also compare the accuracy and running-time performance of our approach against DREBIN [9], Permission-Induced Risk Malware Detection [10], and existing virus scanners. We find that our approach can detect more malware than the other approaches.

1.1 Contributions

1. We develop SigPID, an approach that identifies a subset of permissions (significant permissions) that can be used to effectively identify malware. By using our technique, the number of permissions that needs to be analyzed is reduced by 75%.

2. We evaluate the effectiveness of our approach using only a quarter of the total number of permissions in Android. We find that SigPID can achieve over 90% in precision, recall, accuracy, and F-measure. These results compare favorably with those achieved by an approach that uses all 135 permissions. When we evaluate the malware detection effectiveness of SigPID, we find that our approach is

more effective by detecting 96.54% of malicious apps in the data set while other approaches including Drebin and commercial virus scanners, can only detect 3.99% to 96.41% of malware.

3. To show that the approach can work generically with other supervised learning algorithms, we apply SigPID with commonly used 67 supervised learning algorithms and a much larger dataset (5,494 malicious and 310,926 benign apps). We find that 55 out of 67 algorithms can achieve F-measure of at least 85%, while the average running time can be reduced by 85.6% compared with the baseline approach.

1.2 Organization

The rest of this thesis is organized as follows. Chapter 2 provides the implementation details of the proposed SigPID. Chapter 3 reports the results of our empirical evaluations. Chapter 4 compares our work to other related work. The last Chapter concludes this thesis.

Chapter 2

Introducing SigPID: Significant Permission Identification for Android Malware Detection

The goal of *Significant Permission IDentification* (SigPID) system is to achieve high malware detection accuracy and efficiency while analyzing the smallest number of permissions. To do so, our system extracts permission lists from application packages but instead of focusing on all permissions, SigPID mainly focuses on permissions that can improve the malware detection rate. This, in effect, eliminates the need to analyze permissions that have little or no significant influence on malware detection effectiveness. In a nutshell, SigPID prunes permissions that have low impacts on detection effectiveness using multi-level data pruning, which consists of three major components: (i) permission ranking with negative rate; (ii) permission mining with association rules; and (iii) support based permission ranking. After pruning, SigPID employs supervised machine learning classification methods to identify potential Android malware. Finally, SigPID reports malware detection summary to the analysts.

The complete system architecture of SigPID is shown in Figure 2.1. We then describe the key components in the remainder of this chapter.

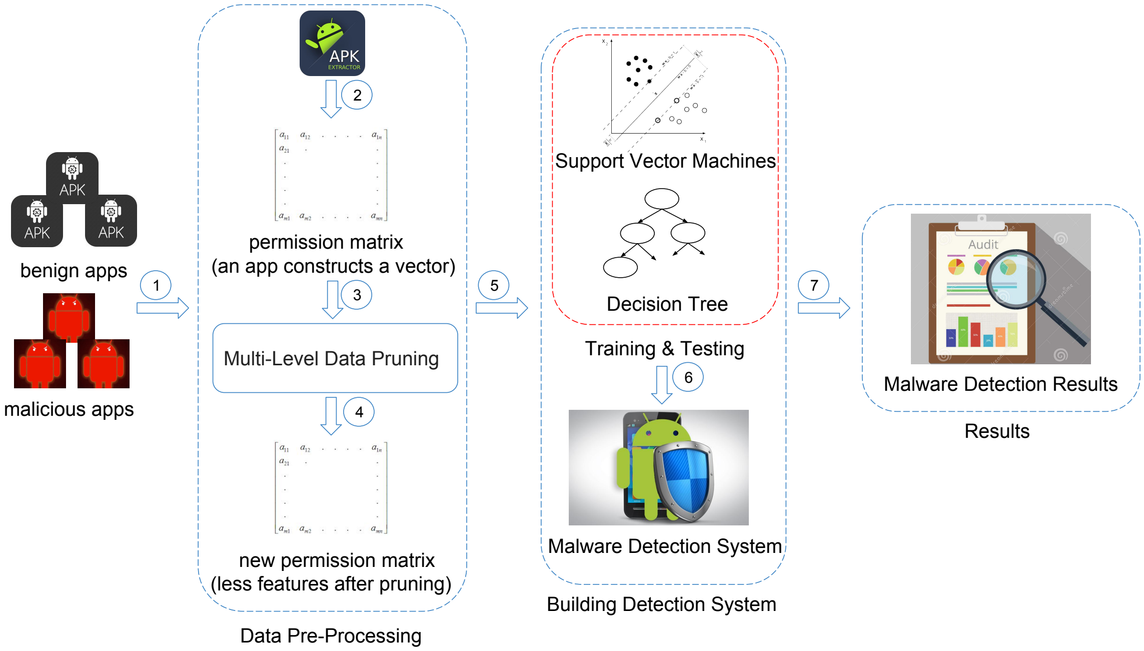

Figure 2.1: System Overview Process including Three Main Parts:Data Pre-Processing, Building Detection System, Detection Analysis

2.1 Multi-Level Data Pruning (MLDP)

A key component of SigPID is the multi-level data pruning process to reduce our permission dataset. Android apps can have up to 135 permissions in total. Most apps do not request all 135 permissions. The ones that an app requests are listed in the apk as part of *manifest.xml*. When we need to analyze a large number of apps (e.g., several hundred thousand), the total number of permissions requested by all apps can be overwhelmingly large, resulting in long analysis time. This high analysis overhead can negatively affect the malware detection efficiency as it reduces analyst productivity.

To address this problem, we use three levels of data pruning methods to filter out

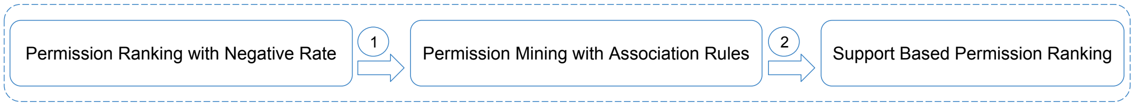
Permission Ranking with Negative Rate
1
Permission Mining with Association Rules
2
Support Based Permission Ranking

Figure 2.2: Multi-Level Data Pruning

permissions that contributed little to the malware detection effectiveness. Thus, they can be safely removed without negatively affecting malware detection accuracy. The complete procedure is illustrated in Figure 2.2. We then describe each level in the pruning process.

2.1.1 Permission Ranking with Negative Rate

Each permission describes a particular operation that an app is allowed to perform. For instance, permission INTERNET indicates whether the app has access to the Internet. Different types of benign apps and malicious apps may request a variety of permissions corresponding to their operational needs. For malicious apps, we hypothesize that their needs may have common subsets. We have seen research efforts that try to use permissions to detect malware [9] [10]. Thus, if they do indeed fall into common subsets, we may not need to analyze all 135 permissions to build an effective malware detection system.

As such, our focus is more on the permissions that create high risk attack surfaces and are frequently requested by malware samples. Our pruning also includes permissions that are rarely requested by the malware so that we can use this information to differentiate between malware and benign apps. At the same time, we also exclude permissions that are commonly used by both benign and malicious apps, as they introduce ambiguity in the malware detection process. For instance, permission INTERNET are frequently requested by both malware and benign apps, as almost all apps will request to access the Internet. Incorporating permission INTERNET for malware detection does not provide any benefits. Therefore, our approach prunes permission INTERNET.

As the next step, we rank permissions based on how they are used by malicious and benign apps. Ranking is not a new concept. Prior work have also used generic permission ranking strategy such as mutual information to identify high risk permissions [10]. However, their approaches tend to only focus on high risk permissions, and ignore no-risk permissions, which are significant permissions in our approach.

Our approach, referred to as *Permission Ranking with Negative Rate* or PRNR, provides a concise ranking and comprehensible results. The approach operates on two matrices, M and B. M represents a list of permissions used by malware samples and B represents a list of permissions used by benign apps. M_{ij} represents whether the j^{th} permission is requested by the i^{th} malware sample, while '1' indicates yes, '0' indicates no. B_{ij} represents whether the j^{th} permission is requested by the i^{th} benign app sample.

Before computing support of permissions from matrices M and B, we first check their sizes. Typically, the number of benign apps (over 300,000 in this thesis) tends to be much larger than the number of malicious apps (5,500 samples in this thesis); as such, the size of B can be much larger than the size of M. With our ranking scheme, we prefer the data set on the two matrices to be more balanced. Training over imbalanced dataset can lead to skewed models [9]. For example, in the case where the size of matrix M is 10 and the size of matrix B is 1000, after applying supervised machine learning techniques for classification, the classification models mainly skew towards capturing the characteristics of matrix B, thereby affecting classification accuracy.

As a result, we first compute their difference using the equation below:

$$\frac{|size(M_j) - size(B_j)|}{min(size(M_j), size(B_j))} < \varepsilon, \tag{1}$$

where $size(M_j)$ represents the number of rows in M and $size(B_j)$ represents the number of rows in B. Variable ε provides the difference in sizes of the two matrices. In this work, ε is very small due to a much larger number of benign apps. The quantity ε can be manually set by policies based on different types of the datasets. In this thesis, we simply set ε to 1.

If we find the difference is smaller than ε (i.e., Eq. 1 holds), PRNR can be implemented using the following equation:

$$R(P_j) = \frac{\sum\limits_i M_{ij} - \sum\limits_i B_{ij}}{\sum\limits_i M_{ij} + \sum\limits_i B_{ij}}, \tag{2}$$

where R(P$_j$) represents the rate of j^{th} permission used for permission ranking.

If we find the difference is bigger than ε (i.e., Eq. 1 does not hold), we need to first balance these two matrices. To do so, we offer two policies. We denote the number of samples in the larger dataset as LN, and the number of samples in the smaller dataset as SN. The first policy randomly selects SN samples from the larger dataset. The second policy uses the equation below to calculate the support of each permission in the larger dataset and then proportionally scales down the corresponding support to match that of the smaller dataset. This can balance the two matrices in spite of their difference in sizes. In the equation 3, variable P_j denotes the j^{th} permission in the case that the number of rows of B is bigger than that of M, we have:

$$S_B(P_j) = \frac{\sum\limits_i B_{ij}}{size(B_j)} * size(M_j), \tag{3}$$

$S_B(P_j)$ represents the support of j^{th} permission in matrix B.

After balancing these two matrices by using eqution 3, PRNR can be implemented using the following modified equation:

$$R(P_j) = \frac{\sum\limits_{i} M_{ij} - S_B(P_j)}{\sum\limits_{i} M_{ij} + S_B(P_j)} \tag{4}$$

The PRNR algorithm is used to perform ranking of our datasets. In the formula above, R(P$_j$) represents the rate of j^{th} permission. The result of R(P$_j$) has a value ranging between [-1, 1]. If R(P$_j$) = 0, this is implies that both benign and malicious apps equally request P$_j$. This also means that P$_j$ has very little impact on malware detection effectiveness. If R(P$_j$) = 1, this means that permission P$_j$ is only used in malicious dataset, which is a high risk permission. If R(P$_j$) = -1, this means that permission P$_j$ is only used in benign dataset which is a low risk permission. Based on the rate of permission with PRNR, we can classify the permissions into two lists: a benign permission list which contains the permissions with negative rates and sorted from -1 to 0, and a malicious permission list which contains permissions with positive rates sorted from 1 to 0. We then remove low impact permissions whose R(P$_j$) are close to 0.

At this point, we have created two sorted permission lists: a benign permission list and a malicious permission list through the use of PRNR. Next, we design a *Permission Incremental System* (*PIS*) to incrementally include permissions based on the order in the two lists. For example, we choose the top permission in the benign list and the top permission in the malicious list as our input features to build malware detection. We then evaluate malware detection by using the following metrics: precision, recall, accuracy, and F-measure. In this thesis, we generate ten sub-datasets by randomly choosing 5,494 benign apps from 310,926 benign apps, so that we can calculate the standard deviation of every performance metric. Next, we choose the top three permissions in both lists to build malware detection. Then, we repeat the process again while increasing the number of top permissions to use for malware detection. The main goal is to find the smallest number of permissions that yields a very similar malware detection effectiveness as that of using the entire data set.

In our thesis, we find that using 66 permissions performs nearly as well as using the whole 135 permissions, as shown in Section 3.2.1. We are able to achieve 91.01% malware detection accuracy while using only 49% of the permissions. Using all 135 permission yields virtually the same effectiveness (90.80%).

2.1.2 Permission Mining with Association Rules

In this section, we introduce the second pruning process in MLDP. After pruning 69 of 135 permissions by using PRNR and PIS, we want to further explore approaches that can reduce non-influential permissions even more. By inspecting the reduced permission list (66 in this case), we find that some permissions always appear together in an app. For example, permission WRITE_SMS and permission READ_SMS are always used together. They both also belong to the malicious permission list. As such, we can associate one, which has higher support, to its partner. This way, one permission can represent both of them. In this example, we can remove permission WRITE_SMS. In order to find permissions that occur together, we apply permission mining with association rules (PMAR).

Data mining with association rules is a method for discovering interesting relations between variables in large databases. For instance, if A and B always occur together, this has high confidence which could be interesting for us. In this thesis, we only consider rules with high confidence, so applying permission mining with association rules only produces a few rules. We employ *Apriori*, a commonly used association mining algorithm, to generate the association rules based on our dataset. Apriori [11] uses a breadth-first search strategy to count the support of itemsets and uses a candidate generation process, which exploits the downward closure property of the support. In this case, we only want to generate the association rules with high confidence even if the permissions have small support values.

Using association rules identified from the permission data, we find 6 rules when we set 98% minimum confidence and 3% as minimum support with Apriori. We remove these 6 permissions from the current dataset of 66 permissions in total. With 60 permissions (55.6% reduction in the number of permissions), we achieve 92.06% detection accuracy, as shown in Section 3.2.2.

2.1.3 Support Based Permission Ranking

At this point, we have reduced the number of permissions needed to perform malware detection by 55%. To further reduce the number of permissions, we turn our focus to the support of each permission. Typically, if the support of a permission is too low, it does not have much impact on malware detection. For instance, we find the permission BIND_TEXT_SERVIC only in benign apps. As such, we may think that any app that uses BIND_TEXT_SERVICE is benign. However, this permissions is used only by one app out of over 300,000 benign apps. As such, only relying on the rate provided by PRNR is inaccurate. We also need to prune out permissions with low support.

To prune out permissions with low support, one obvious option is to rank permissions based on support and then use PIS to find the least number of permission that yields high accuracy. However, there is an approach based on using a support threshold that involves less complexity and can result in shorter analysis time. To do so, we set a support threshold γ, and any support value for a permission that is smaller than γ would be pruned. A large value of γ would allow us to prune more permissions, but it also reduces accuracy and F-measure. A small value of γ would maintain more permissions, resulting in higher runtime overhead. To balance the accuracy and the efficiency, we experiment with different values and find that 0.5% yields a good balance, so it is used as the threshold in this thesis.

2.2 Malware Detection using Significant Permissions

We first use SVM and a small dataset to test our MLDP model from the previous section. SVM determines a hyperplane that separates both classes with a maximal margin based on the training dataset that includes benign and malicious application. In this case, one class is associated with malware, and the other class is associated with benign apps. Then, we assume the testing data as unknown apps which will be classified by mapping the data to the vector space to decide whether it is on the malicious or benign side of the hyperplane. Then, we can compare all analysis results with their original records to evaluate the malware detection correctness of the proposed model by using SVM.

In order to show applicability and scalability of MLDP, we employ 67 commonly known machine learning algorithms and enlarge our dataset. We compare the results between malware detection rate using all 135 permissions (baseline) and malware detection using MLDP for each supervised machine learning algorithms, as shown in Section 3.3. We observe that machine learning algorithms with a tree structure usually build better malware detection compared to others. However, they also tend to take more time and memory. Consequently, it is more advantageous to use MLDP to perform malware detection as it can be as effective while conserving time and memory. We will provide more details in the next section.

2.3 Advanced MLDP with Fusion of Multiple Lists and X-value

There are two obvious inefficiency of SIGPID when we use MLDP for significant permissions. First, there are two ranking lists as the system implements two PIS in MLDP. Processing each PIS can take a significant amount of time. Second, we do not know the least number of permissions that can help us to build a good malware detection system when we implement PIS. Finding the least number of permissions would enable us to build a high performance malware detection system based on PIS.

In order address these two short-comings of MDLP, we introduce a new algorithm called FML (Fusion of Multiple Lists) and a new attribute called X-value. Next, we describe them in turn.

2.3.1 Fusion of Multiple Lists

The first algorithm fuses multiple lists into one list before ranking each of them in MLDP. Fusion of Multiple Lists (FML) allows us to apply PIS only once to the resulting list instead of twice as in MLDP prior to applying this optimization. Since processing PIS is a time consuming process, this optimization can make approach more efficient. The fusion process contains three steps.

Step 1: The algorithm combines the positive rate and negative rate lists. After it calculates the $R(P_j)$, the rate of each permission falls between -1 and 1. However, both extreme values are significant to classify applications, as such, we can simplify these values by considering the absolute value of all rates $R(P_j)$ for each permission. Then, our algorithm calculates the maximum value, minimum value, median value, mean value, and standard deviation of each list. If it follows the normal distribution such as the positive-rate list, we prune the dataset in the first step by removing any permission whose rate is less than $mean - standard\ deviation$. If the list does not follow the normal distribution such as the frequency list, our algorithm removes any permission, which has no contributions. For example, we have total about 11,000 applications for building malware detection system. If some permissions are used fewer than 11 times in all apps, whose support is less than 0.1% (α), it should be removed. **Step 2:** After data pruning in the first step, we normalize the scores of each list to the maximum value. This causes each updated score to be in the range of 0 and 1. Next, the algorithm sorts the remaining permissions alphabetically and put the normalized scores of each permission on the two lists in a table (e.g., Permission, Normalized Score from Positive List, Normalized Score from Frequency List).

Step 3: Choose a function for multiple normalized lists. We create a table in **Step 2**, and each permission has two normalized scores from positive-rate list and frequency list. In the table, we use $S1_i$ represent the normalized score from positive-rate list for i_{th} permission, and $S2_i$ represent the normalized score from frequency list for i_{th} permission. We need to calculate a new score which can represent two scores and be used in the fusion list. As a result, here we choose to use math function to calculate a new score, where S_i is a new score for i_{th} permission. There are multiple approaches to calculate a new score, each way also causes different scores in the new fusion list. To identify a suitable approach, we experiment with three functions described next.

$$S_i = (S1_i + S2_i)/2, \tag{5}$$

Function 5 considers the total score of $S1_i$ and $S2_i$ for i_{th} permission. Although $S1_i$ is a small score, i_{th} permission still can be considered as a significant permission when $S2_i$ is a very large score.

$$S_i = S1_i * S2_i, \tag{6}$$

Function 6 considers the multiple score of all normalized score in multiple lists. For example, when $S1_i = 1$, $S2_i = 0.1$, $S1_j = 0.5$ and $S2_i = 0.5$, S_i is higher than S_j if we choose to use Function 5 and S_j is higher than S_i if we use Function 6.

$$S_i = 1/2 * (S1_i + S2_i) * (S1_i * S2_i), \tag{7}$$

Function 7 combines multiple score and the total score of all normalized scores in multiple lists produce the best result.

2.3.2 X-value

We introduce X-value to find the the least number of permission for PIS. X-value is the average distance between recall to f-measure and precision to f-measure which can help us identify the least number of permissions that returns a high and stable f-measure.

When we process PIS in the two datasets, it always shows similar precision, recall and f-measure. Precision initially decreases, and then subsequently increases. Recall initially increases then decreases. At some point, precision, recall and f-measure are very close and then f-measure subsequently becomes stable meaning changes occur in small ranges.

Based on an empirical evaluation, at some point X-value approaches 0, meaning all three values are almost the same. Thus, we can set a small value β. If $X - value$

is smaller than β, this is a point where f-measure is stable. As such, X-value helps us to identify the best point to stop PIS and also the least permissions that can be used for for malware detection.

Table 2.1: Rankings by PRNR

Permission Ranking with Negative Rate	
Malicious List Top 33	Benign List Top 33
RECEIVE_WAP_PUSH	READ_CALL_LOG
WRITE_APN_SETTINGS	WRITE_CALL_LOG
WRITE_SMS	NFC
READ_SMS	SET_ALARM
BROADCAST_WAP_PUSH	READ_PROFILE
DELETE_PACKAGES	READ_USER_DICTIONARY
BROADCAST_PACKAGE_REMOVED	WRITE_USER_DICTIONARY
RECEIVE_MMS	SET_TIME
INSTALL_PACKAGES	WRITE_PROFILE
BRICK	BIND_DEVICE_ADMIN
ADD_SYSTEM_SERVICE	SET_PROCESS_FOREGROUND
EXPAND_STATUS_BAR	BIND_REMOTEVIEWS
SET_PROCESS_LIMIT	BIND_ACCESSIBILITY_SERVICE
RECEIVE_SMS	WRITE_SOCIAL_STREAM
SEND_SMS	READ_SOCIAL_STREAM
SET_WALLPAPER_HINTS	ADD_VOICEMAIL
DISABLE_KEYGUARD	BIND_VPN_SERVICE
FACTORY_TEST	SET_POINTER_SPEED
RESTART_PACKAGES	BIND_TEXT_SERVICE
BIND_APPWIDGET	USE_CREDENTIALS
MODIFY_PHONE_STATE	MANAGE_ACCOUNTS
INTERNAL_SYSTEM_WINDOW	CHANGE_COMPONENT_ENABLED_STATE
DEVICE_POWER	ACCESS_MOCK_LOCATION
PERSISTENT_ACTIVITY	AUTHENTICATE_ACCOUNTS
WRITE_CONTACTS	CAMERA
SET_ALWAYS_FINISH	CHANGE_WIFI_MULTICAST_STATE
PROCESS_OUTGOING_CALLS	READ_EXTERNAL_STORAGE
CHANGE_WIFI_STATE	READ_CALENDAR
BROADCAST_SMS	FLASHLIGHT
READ_FRAME_BUFFER	READ_SYNC_STATS
READ_LOGS	GET_ACCOUNTS
DELETE_CACHE_FILES	CLEAR_APP_USER_DATA
STATUS_BAR	BROADCAST_STICKY

Table 2.2: Rankings by PRNR

Mutual Information	
Top 66	
READ_SMS	WRITE_CALL_LOG
WRITE_SMS	VIBRATE
SEND_SMS	CHANGE_NETWORK_STATE
WRITE_APN_SETTINGS	DEVICE_POWER
RECEIVE_SMS	WRITE_SETTINGS
INSTALL_PACKAGES	ADD_SYSTEM_SERVICE
READ_PHONE_STATE	ACCESS_NETWORK_STATE
READ_EXTERNAL_STORAGE	ACCESS_LOCATION_EXTRA_COMMANDS
RESTART_PACKAGES	WAKE_LOCK
RECEIVE_BOOT_COMPLETED	ACCESS_COARSE_LOCATION
WRITE_CONTACTS	GET_ACCOUNTS
READ_CONTACTS	BROADCAST_PACKAGE_REMOVED
CHANGE_WIFI_STATE	WRITE_OWNER_DATA
ACCESS_WIFI_STATE	BROADCAST_WAP_PUSH
DISABLE_KEYGUARD	ACCESS_MOCK_LOCATION
DELETE_PACKAGES	WRITE_SYNC_SETTINGS
READ_LOGS	USE_CREDENTIALS
CALL_PHONE	WRITE_SECURE_SETTINGS
RECEIVE_WAP_PUSH	DELETE_CACHE_FILES
RECEIVE_MMS	READ_CALENDAR
READ_HISTORY_BOOKMARKS	PERSISTENT_ACTIVITY
EXPAND_STATUS_BAR	GET_PACKAGE_SIZE
PROCESS_OUTGOING_CALLS	STATUS_BAR
READ_CALL_LOG	BROADCAST_SMS
INTERNET	FLASHLIGHT
WRITE_HISTORY_BOOKMARKS	BIND_APPWIDGET
GET_TASKS	CHANGE_CONFIGURATION
MODIFY_PHONE_STATE	INTERNAL_SYSTEM_WINDOW
SET_WALLPAPER_HINTS	MANAGE_ACCOUNTS
WRITE_EXTERNAL_STORAGE	READ_FRAME_BUFFER
SET_WALLPAPER	REORDER_TASKS
CAMERA	SET_PROCESS_LIMIT
MOUNT_UNMOUNT_FILESYSTEMS	SET_PREFERRED_APPLICATIONS

Chapter 3

Evaluation

In this chapter, we evaluate the malware detection effectiveness of the SigPID system. Our evaluation employs 10,988 apps (5,494 malware, 5,494 benign apps from 310926 benign apps) to build our malware detector. Through pruning, our system identifies only 34 significant permissions (reduction of 74.8%) and when only these permissions are used to detect malware, our system achieves 91.9% accuracy. Next, we list the major results identified through our performance evaluation.

Table 3.1: Results with Multi-Level Data Pruning

Number of Features	Status	Recall	Precision	Accuracy	F-measure	STD
135	Orignial	82.16%	99.33%	90.80%	89.93%	0.008640585
66	PRNR	87.95%	93.93%	91.01%	90.75%	0.012360538
60	PMAR	86.95%	96.93%	92.09%	91.66%	0.008236407
34	FPR	87.67%	95.98%	91.94%	91.59%	0.009718129

1. Multi-Level Data Pruning Effectiveness. Multi-level data pruning consists of three main components: permission ranking with negative rate, permission mining with association rules, and support based permission ranking. We evaluate the malware detection performance by enabling these multiple levels sequentially to verify the performance improvement contribute by each level of

permission mining procedure. In addition, we also evaluate the runtime efficiency of multi-level data pruning. SVM algorithm is employed to perform malware detection.

2. Malware Detection Effectiveness with Different Machine Learning Algorithms. We apply SigPID over multiple machine learning classification algorithms to validate the wide applicability of SigPID mechanism.
3. Comparison with Other Approaches. We compare the classification effectiveness of SigPID with results of approaches using other permission ranking methods such as Mutual Information [10].

3.1 Data Set

In this section, we illustrate the process of building a permission dataset. In total, we have 5,494 malicious apps and 310,926 benign apps downloaded from Google play store in June 2013 [9]. The malicious apps are classified into 178 families, and the benign apps are grouped into a single family. Then, the requested permission list is built by extracting permission requests from each app listed in AndroidManifest file. The permission information is translated into a binary format dataset where '1' indicates that the app requests the permission, and '0' indicates the opposite. The permission lists extracted from malicious apps and benign apps are combined to form a holistic dataset for data analysis.

3.2 Evaluating Multi-Level Data Pruning

In this section, we report the effectiveness of each component of multi-level data pruning. First, we evaluate the effectiveness of detection system after enabling permission ranking with negative rate. We also implement another permission ranking algorithm based on Mutual Information [10]. Second, we evaluate the detection effectiveness after enabling permission mining with association rules. Last, we evaluate the detection effectiveness after enabling support based permission ranking. Finally, we combine all three approaches to evaluate the effectiveness of multi-level data pruning.

3.2.1 Evaluating Permission Ranking with Negative Rate

First, we implement PRNR to derive both the malicious permission ranking list and benign permission ranking list. Then, we implement the PIS to incrementally include permissions based on the ranking lists. Two permissions from both lists are added to the significant permission list for each round, the results of which are illustrated in the Fig. 3.1 and Fig. 3.2.

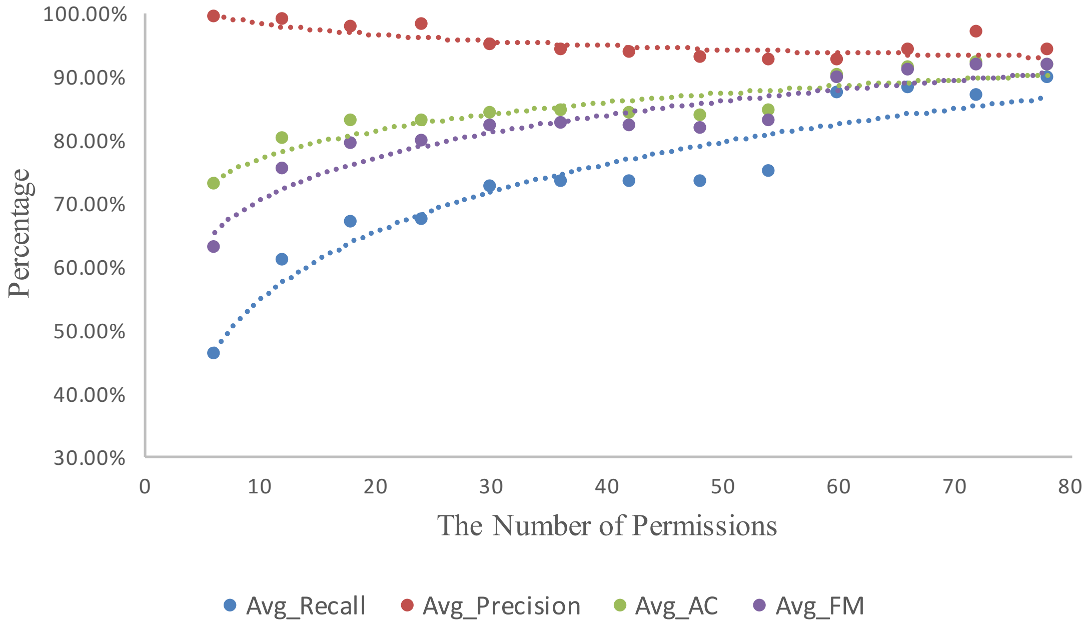

Figure 3.1: Malware Classification Performance of Permission Incremental System

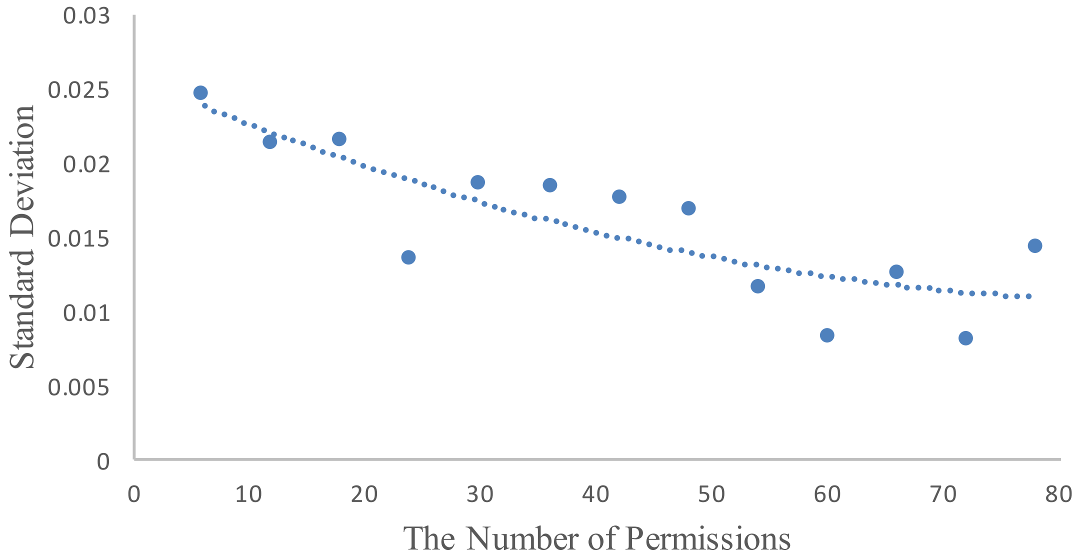

Figure 3.2: Standard Deviation with Incremental Number of Permissions

According to Fig. 3.1 and Fig. 3.2, with an increasing number of permissions, the classification accuracy, recall and F-measure are improving every round. Meanwhile, the precision slightly degrades every round, but always stays above 95% as shown in Fig. 3.1. One keen observation is that the recall, accuracy and F-measure plateau after the number of permissions reaches 66. Fig. 3.2 also shows that the standard deviation of F-measure is low when the number of permissions reaches 66, i.e., the malware detection effectiveness remains the same with different datasets. We take the standard deviation of F-measure, since F-measure is a more aggregated measure of both precision and recall, the stabilization of which better represents that of the detection system.

Table 3.2: Results of Using 67 Machine Learning Algorithms

Number of Features	Precision	Recall	F_measure	ROC	Training	Testing	Total
					(Seconds)		
34	83.10%	95.51%	87.97%	88.83%	0.77	2.05	2.82
135	84.78%	96.70%	89.44%	89.73%	4.16	15.31	19.47

Fig 3.1 further shows that, after 66 permissions, recall stays above 87%, and precision stays over 93%. Our malware detection system can accurately classify both benign apps and malicious apps well when employing only 66 permissions.

In Table **??**, we list the 66 permissions in the malicious permission list and benign permission list. As a comparison, we also list the top 66 permissions using another permission ranking method called Mutual Information [10]. Different permission ranking methods induce different ranking lists. For instance, using PRNR, we drop the permission INTERNET since it shows that both benign and malicious apps often need INTERNET. However, mutual information based ranking method keeps the permission INTERNET in the list as permission INTERNET is frequently requested by all apps. Therefore, we believe our algorithm can retain more significant permissions by pruning less important or meaningless permissions compared with other permission ranking methods.

After applying the PRNR method, we evaluate the model and compare the results with the model using 135 permissions. In Table 3.1, we can find that both methods achieve nearly 90% accuracy and F-measure. However, by inspecting the results in detail, we find that the new malware detection model with 66 permissions has a higher recall rate, and a lower precision rate than the original detection model with 135 permissions. In other words, the new model detects more malware correctly, but at the same time, it slightly sacrifices accuracy for classifying benign apps. But in terms of accuracy and F-measure, the new model presents slightly better results. As such, the new malware detection model with PRNR ranking method is as effective as the model that uses all permissions.

3.2.2 Evaluating Permission Mining with Association Rules

Next, we implement permission mining with association rules to perform the second layer of permission mining. Here, we find 6 rules satisfying our associative requirements when we set our confidence level to 95%. After pruning 6 more permission features in the dataset, we only retain 60 features. In Table 3.1, we can see that the new detection model achieves above 90% accuracy and F-measure. Looking into the details, we observe that higher precision is achieved with the new model employing fewer permissions. The association rules mining shows that permission WRITE_SMS and permission READ_SMS have a 99.5907% chance to appear together. Meanwhile, permission WRITE_SMS is very frequently used in both malware and benign applications while READ_SMS is used mainly by malware. When we remove the permission WRITE_SMS, only applications requesting permission READ_SMS are classified as malware.

3.2.3 Evaluating Support based Permission Ranking

After pruning the dataset with PRNR and PMAR, we continue to improve the significant ranking list with support based permission ranking (SPR). Support based permission ranking relies on the fact that some permissions are rarely requested by apps in the dataset, which can be pruned accordingly. Because different datasets contain different attributes in terms of permissions, we need to set different thresholds to prune different datasets. We can implement PIS based on support ranking to find the least requested permissions that can be used to set the threshold. However, PIS method brings additional computational overhead. Therefore, here we manually set the threshold to 0.5%, which is identified based on an empirical evaluation that shows good results for our dataset.

Table 3.3: Results with Top 5 Machine Learning Algorithms (Normalized)

Name of Algorithm	Precision	Recall	F_measure	ROC
RandomForest	93.31%	95.63%	94.46%	0.98032402
PART	92.45%	96.54%	94.45%	0.973974003
FT	92.04%	96.78%	94.35%	0.960329922
RandomCommittee	92.73%	95.93%	94.30%	0.979415481
RotationForest	92.03%	96.66%	94.29%	0.978340985

After pruning the dataset with SPR, we only retain 34 significant permission features based on the dataset. In Table 3.1, we show that the detection model with 34 permissions still achieves above 90% accuracy and F-measure.

Note that the pruning order of MLDP can be rearranged. Based on our results, we find that both PNRN and SPR can prune more permissions. As such, we can use these two levels first. After that, we can run the PMAR further to refine the significant permission list, as it takes more time to build association rules with more permissions in the dataset.

3.3 Evaluating Malware Detection Efficiencies with Different Machine Learning Algorithms

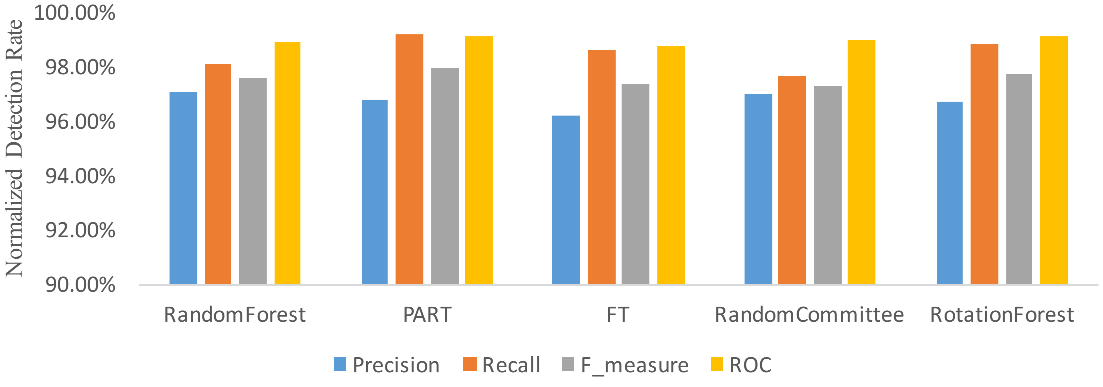

Figure 3.3: Results with Top 5 Machine Learning Algorithms

In order to show the generality of MLDP, we present different malware detection models using different machine learning algorithms in Weka [12]. We experiment with 67 different machine learning algorithms with both the original dataset and the dataset after data pruning using MLDP. We want to evaluate the performance of MLDP in any general algorithm in terms of detection accuracy and running time performance. In Table 3.2, we have the average results of different machine learning algorithms using 34 and 135 permissions, respectively. There is only a minor performance difference between the case with 34 permissions and the case with 135 permissions. However, by examining the running time, the models with 34 permissions only take 2.8 seconds on average, compared to 19.46 seconds from the malware detection model using 135 permissions, which translates into an 85.6% improvement. The detection model with 135 permissions takes almost 7 times as long to run compared to the detection model with MLDP.

Here, we use precision, recall, F-measure and ROC to evaluate the malware detection model. In Figure 3.3, we normalize the detection results with 34 permissions by the performance of the model with 135 permissions. We can see that although the malware detection model with full permission list can achieve better performance than the detection model with 34 permissions in terms of the precision, recall, F-measure, and ROC, the difference is very small. Table 3.3 reports the top 5 malware detection models that achieve above 94% F-measure, and above 92% in other performance measures, thereby proving the detection model is still capable of classifying the malware and benign apps with high accuracy.

Figure 3.4 reports the execution time of using 34 permissions, normalized to the execution time of the model with 135 permissions. The new malware detection model consumes about half the time of the model with a complete permission list. Meanwhile, less features in the dataset also can save lots of memory space. Surprisingly, if we use

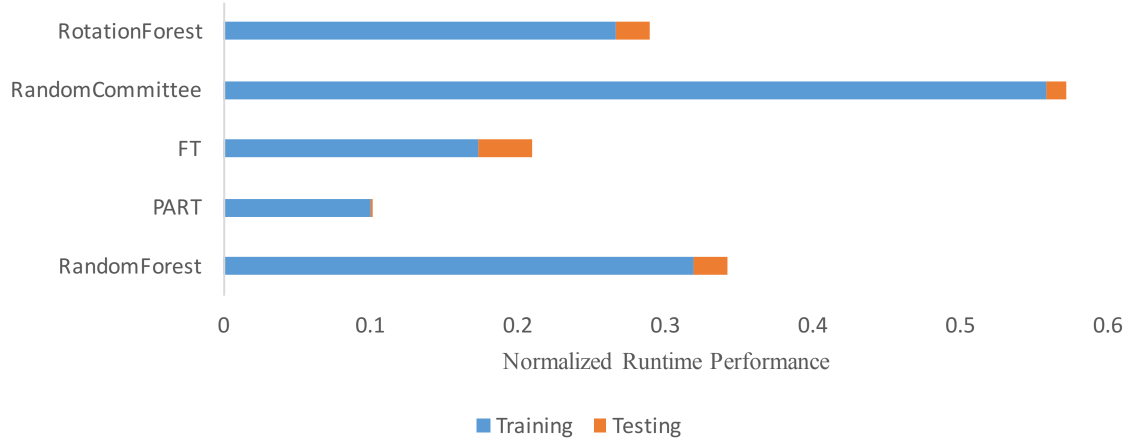

Figure 3.4: Runtime Performance with Top 5 Machine Learning Algorithms

PART algorithm [13], more than 90% time can be saved comparing to the malware detection model with the complete permission list.

Based on the tested 67 machine learning algorithms, we also find that the machine learning methods based on tree structure can produce better results. Among the top 10 efficient algorithms, 5 are designed with tree structures. The tree structure based method usually takes a large amount of space and time to run the classification process, so our MLDP can serve as a solution to help significantly improve running time efficiency of the malware detection model based on tree structures.

3.4 Improvement of MLDP

3.4.1 Scalability of MLDP

We have shown that MLDP can work well when we used 1,661 malicious apps and a selection of 1,661 apps from a corpus of 310,926 benign apps. We discovered that 34 out of 135 permissions are significant (a reduction of 74.8%). When we used these 34 permissions to detect malware, our system achieves 91.9% accuracy. Our proposed approach performed better in terms of both f-measure and accuracy than using the full permission set. In order to show scalability of SIGPID, we enlarge our data set by roughly 3 times; that is, the new dataset contains 5,494 malicious apps and 5,494 from 310,926 benign apps. Note that we use PIS in both PRNR and SPR.

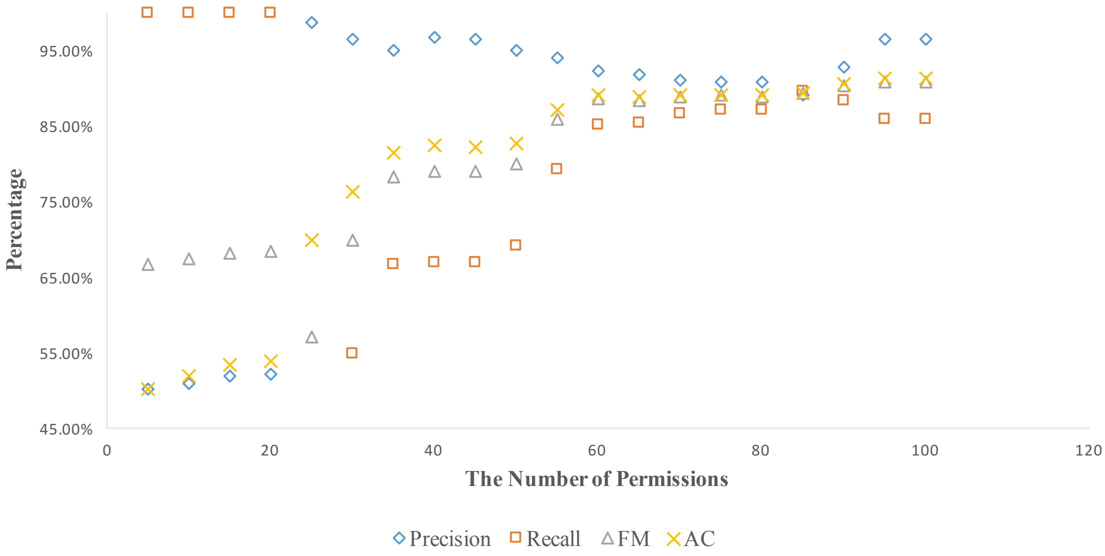

Figure 3.5: First Step: Malware Classification Performance of PIS in PRNR

From above Fig. 3.5, at least 90 permissions are needed to build a stable system based on PRNR. This is different than when we used a dataset containing 1,661 malicious apps. In that scenario, we need only 66 permissions to build a stable system based on PRNR. This difference is expected as different datasets should result in different points to stop PIS.

From Fig. 3.6 , the graph becomes stable very soon after we finish the SPR. We only need 15 permissions to build a stable system. We also provide Table 3.4 to report f-measure and accuracy in greater details.

We can achieve the best f-measure when we use 25 permission in the second PIS with SPR. We can also achieve the best accuracy by using 40 permissions with SPR. Note that when we use threshold ε which use 34 permissions, we get comparable results.

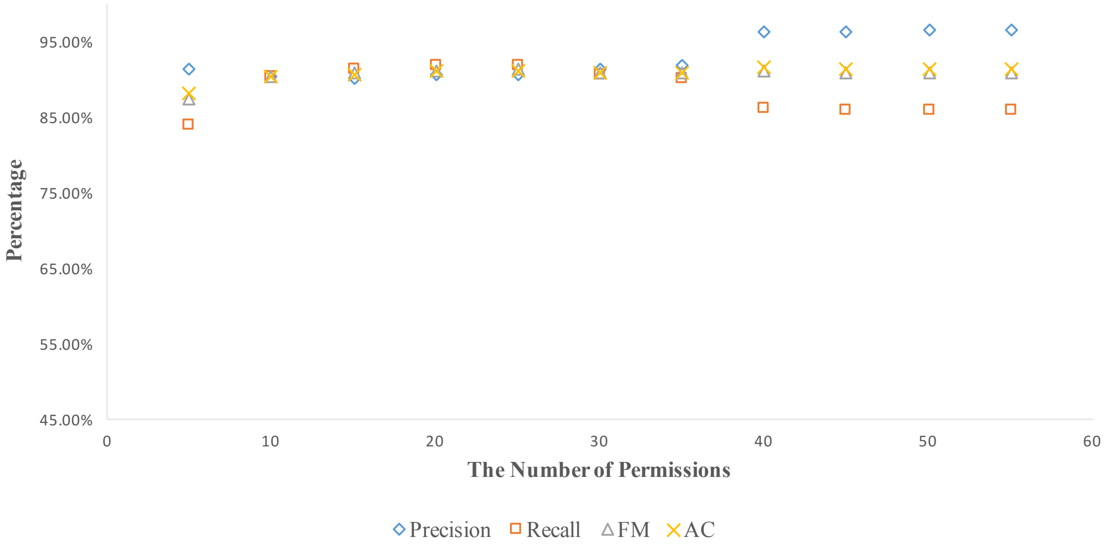

Figure 3.6: Second Step: Malware Classification Performance of PIS in SPR

Table 3.4: Malware Classification Performance of PIS in SPR

Num_of_Feature	Precision	Recall	FM	ACC	X-value
5	91.29%	83.90%	87.44%	87.94%	0.073838409
10	90.21%	90.24%	90.22%	90.22%	0.000292543
15	90.12%	91.21%	90.66%	90.61%	0.010895847
20	90.47%	91.65%	91.05%	91.00%	0.011831152
25	90.64%	91.77%	91.17%	91.10%	0.01127205
30	91.27%	90.58%	90.82%	90.82%	0.006842435
35	91.83%	90.05%	90.80%	90.86%	0.017821964
40	96.28%	86.19%	90.96%	91.43%	0.100880652
45	96.28%	85.94%	90.82%	91.31%	0.103398698
50	96.34%	85.82%	90.77%	91.28%	0.105208413
55	96.35%	85.80%	90.77%	91.27%	0.105515343
135	98.81%	83.73%	90.65%	91.36%	0.150770346

As such, regardless whether PIS or a threshold is used, the number of significant permissions is much smaller than 135, which is the total number of permissions.

3.4.2 Algorithm: Fusion of Multiple Lists

In our MLDP, we use two rank lists, so we need to apply PIS twice to find the significant permissions. Because PIS is rather an expensive process, one option, as described in Chapter 2, is to fuse multiple lists into one list and then apply PIS just once. Table 3.5 reports the basic statistics of the normalized positive rate list and frequency list.

Table 3.5: Statistics Analysis For Normalized Positive Rate List and Frequency List

	MAX	MIN	AVG	MED	STD
Positive Rate	1	0.242818576	0.640458489	0.659144017	0.220992305
Frequency	1	0.001392218	0.084424815	0.025134806	0.154414719

From Table 3.5, after we take the absolute values of values in the negative rate list, we can remove permissions with low scores. A low score is defined by $mean - standard\ deviation$. The value of the normalized positive list follows the normal distribution. However, values in the frequency list do not follow the normal distribution. As shown in Fig 3.7, apps frequently used about 40% of permissions. As such, we can not use normal distribution to remove insignificant permissions. Instead, we use a very small threshold as 0.1% to remove the permissions with low usage frequency. The usage frequency is defined as how many time a permission is used divided by the number of apps in the corpus.

Next, we need to calculate the new score for each permission. As mentioned earlier, we experiment with using three functions (Function 5 to Function 7. Table 3.6 reports the top 40 permissions with RANK1 with function 5, and RANK3 with function 7. As shown, the rankings are different. Specifically, NFC is considered as the most significant permission by Function 5. While it has a high positive rate, it also has a very low frequency (0.15%). This means that is is not significant. This table also shows that Function 7 produces a more meaningful significant permission list.

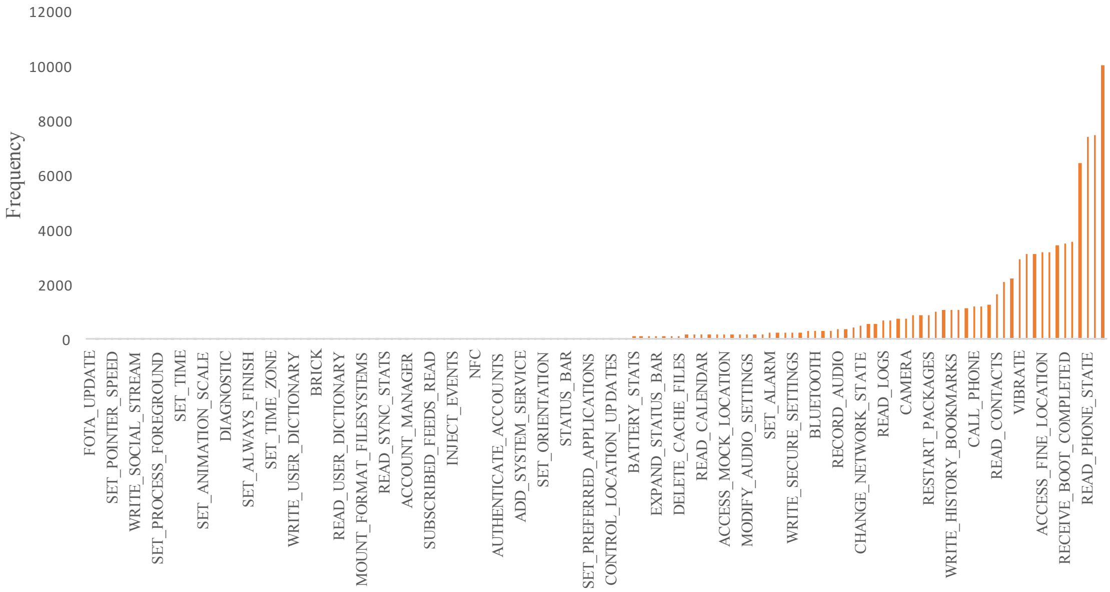

Figure 3.7: Frequency of Permissions

After building a new list, we need to rank permissions by the new scores and try PIS with the new rank list. Fig 3.8 shows the new results with PIS.

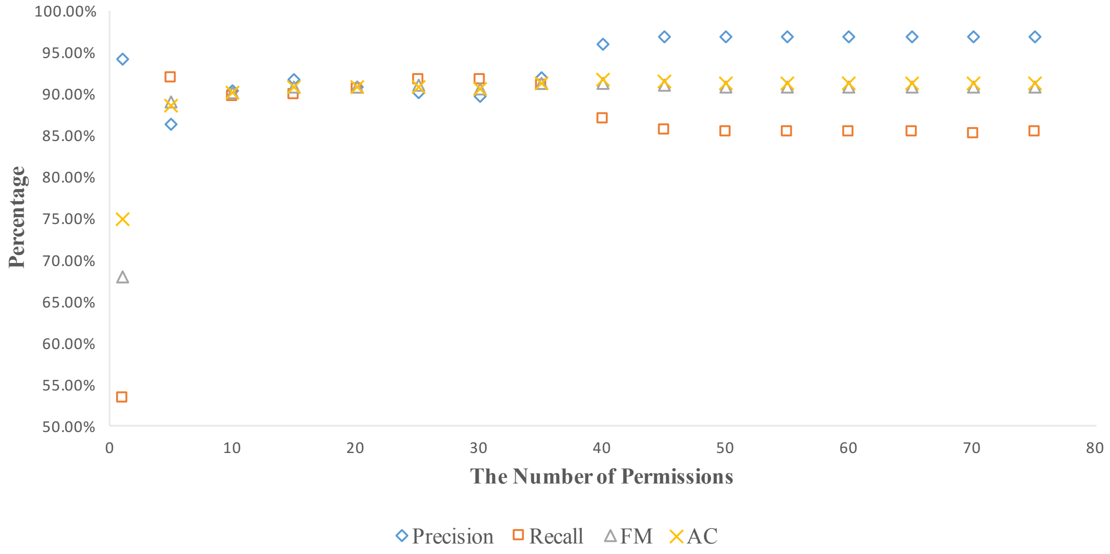

Figure 3.8: Malware Classification Performance of PIS with FML

Fig 3.8 shows that we can build a stable and high performance malware detection system as little as 15 permissions. The PIS stopping point is reached much sooner than when we use the permission ranking list with negative rates. This can lead to significant time savings. Also note that with 15 significant permissions, we can achieve 90.67% f-measure, which is higher than 90.65% with using 135 permissions. It also achieves nearly the same results as using the top 40 permissions, which achieves the best results in both f-measure(91.12%) and accuracy(91.54%).

3.4.3 X-value

We then introduce X-value as a predictive maeasure to indicate that f-measure is about to be stable. X-value represents the average distance between precision to f-measure and recall to f-measure. When X-value is close to 0, it means precision, recall and f-measure are very close. When they are very close, the PIS stopping point should be near.

Fig. 3.9 and Fig. 3.10 is the X-values from Fig. 3.5 and Fig. 3.6.

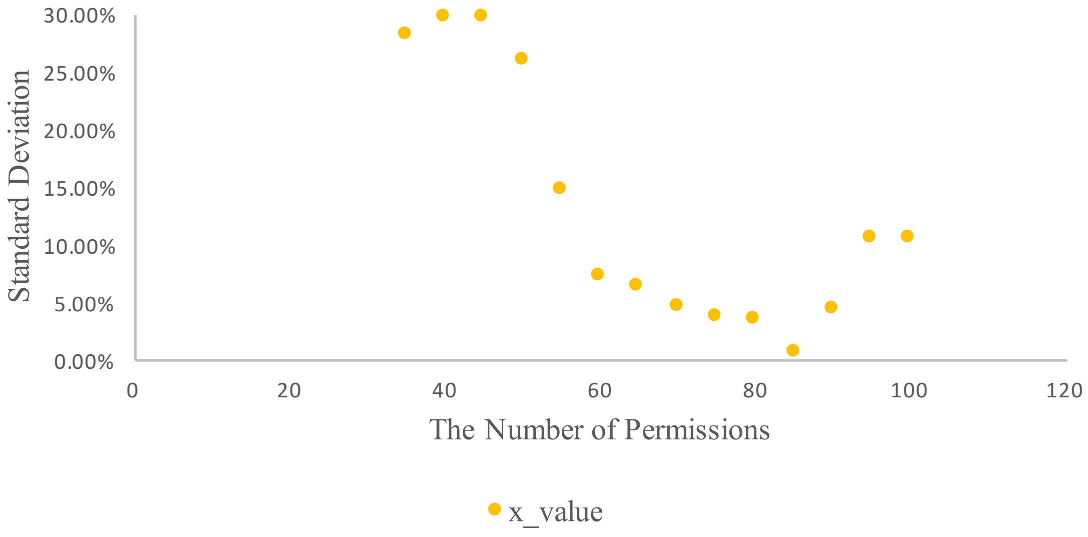

Figure 3.9: X-value in PIS with PRNR

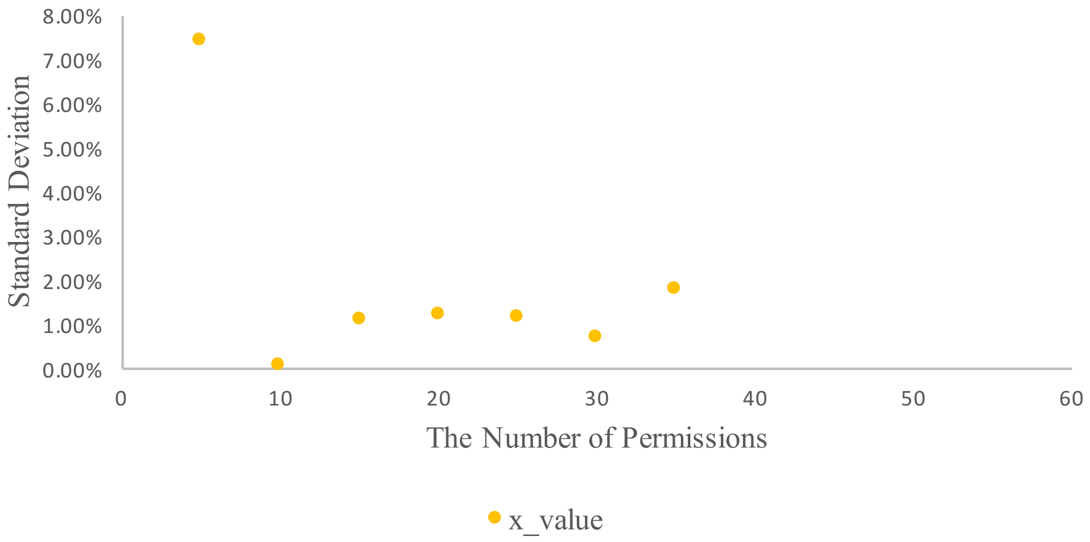

Figure 3.10: X-value in PIS with SPR

From Fig. 3.9, PIS with 85 permission holds lowest X-value. Also shown, f-measure becomes stable with 90 permissions. From Fig. 3.10, PIS with 10 permission holds lowest X-value, and the f-measure is stable with least 15 permissions. Then, we can decide a small threshold β. If $X - value\ smaller\ than\ \beta$, we only need to perform a few more iterations of PIS (usually no more than 3 more iterations), and pick the biggest f-measure as our best results.

Based on an empirical evaluation, we set the 0.01 as our threshold. For example, in PIS with SPR, we find the 10 permissions is the stop points. Then we continue to calculate the models with 15 permissions (first iteration), 20 permissions (second iteration) and then 25 permissions (third iteration). Then, we find that using 25 permissions returns the best results of 91.05% f-measure and 91.00% accuracy. We also experiment with using more than three iterations but find no improvements after three iterations.

3.5 Comparison with Other Approaches

In this section, we compare our detection results with other malware detection approaches, listed as follows:

- Drebin [9] is an approach that uses static analysis to build data set based on permissions and other features from apps. It then utilizes Support Vector Machine (SVM) algorithm to classify malware dataset.
- Permission-induced Risk Malware Detection [10] is an approach that applies permission ranking, such as mutual information. They use the permission ranking and choose the top 40 risky permissions for malware detection.

The comparison results are shown in Table 3.7. Drebin uses more features than our SigPID, including API calls and network addresses. As a result, Drebin outperforms PermissionClassifier in detection accuracy. We also compare the results against 10 existing anti-virus scanners [9].

In addition, we also evaluate the mutual information based permission ranking method [10]. The malware detection with only Mutual Information, produces 82.08% recall, 99.53% precision, and around 90% for both accuracy and F-measure. The mutual information based ranking method only takes risky permissions into account which might miss some of the important differentiable permissions. In addition, many insignificant permissions will still be retained for such permission-based malware detection method. As mentioned previously, permission INTERNET would be considered as a risky permission in Mutual Information, which is actually an insignificant permission as shown by our results.

Table 3.7 shows that when we combine SigPID with PART, we can outperform all other detection tools using only 34 permissions. On the other hand, the permission lists used by Drebin contain many meaningless features, and performance improvements can be achieved by integrating SigPID with PART into Drebin to improve both malware detection accuracy and running time performance. We will explore this integration in our future work.

Discussion. Compared to other permission rankings that only consider risky permissions, SigPID considers a broader criteria that also include non-risky permissions (e.g., READ_CALL_LOG), which is only used in benign apps and has a high support of the whole dataset. We call the risky and non-risky permissions with high support *significant permissions*, allowing SigPID to be more effective at distinguishing between malicious and benign apps than results reported by previous work.

We also see that, despite a small number of permissions, our approach outperforms most of existing malware scanner available today. This is because most of these techniques rely on signature matchings; so if a type of malware signatures is not available, the system would not be able to detect that particular type. We also show that our approach is more effective than DREBIN when we combine our permission pruning with PART. DREBIN is a more complex malware detection approach that also uses static program analysis. We plan to also explore a combination of using static program analysis with SigPID to assess whether we can achieve higher detection effectiveness.

In this work, we only consider one permission ranking algorithm at a time. It is possible that a composite ranking schemes that combines two or more algorithms may provide more power to identify malware while using even fewer significant permissions without losing its effectiveness. In addition, we observe that permission rankings can prune more permissions with little negative impacts on malware detection. As such, we think applying permission rankings before permission mining with association rules can potentially improve the efficiency. Lastly, it would be quite useful to provide a reporting mechanism so that users can better comprehend the results. An example may include a report of permission ranking in sorted orders. We plan to explore these research ideas as part of future work.

Table 3.6: RANK1 VS RANK3

RANK1_Avg	RANK3_Combined
READ_PHONE_STATE	SEND_SMS
SEND_SMS	READ_PHONE_STATE
READ_SMS	READ_EXTERNAL_STORAGE
READ_EXTERNAL_STORAGE	READ_SMS
RECEIVE_SMS	RECEIVE_SMS
WRITE_SMS	RECEIVE_BOOT_COMPLETED
READ_CALL_LOG	WRITE_SMS
WRITE_CALL_LOG	ACCESS_WIFI_STATE
WRITE_APN_SETTINGS	INSTALL_PACKAGES
INSTALL_PACKAGES	READ_HISTORY_BOOKMARKS
SET_ALARM	READ_CONTACTS
RECEIVE_WAP_PUSH	CHANGE_WIFI_STATE
RECEIVE_BOOT_COMPLETED	WRITE_HISTORY_BOOKMARKS
USE_CREDENTIALS	WAKE_LOCK
RECEIVE_MMS	WRITE_APN_SETTINGS
DELETE_PACKAGES	RESTART_PACKAGES
WRITE_SECURE_SETTINGS	READ_CALL_LOG
AUTHENTICATE_ACCOUNTS	WRITE_SETTINGS
DELETE_CACHE_FILES	DISABLE_KEYGUARD
NFC	CHANGE_NETWORK_STATE
BROADCAST_WAP_PUSH	GET_TASKS
UPDATE_DEVICE_STATS	READ_LOGS
SUBSCRIBED_FEEDS_READ	DELETE_PACKAGES
GET_PACKAGE_SIZE	SYSTEM_ALERT_WINDOW
READ_HISTORY_BOOKMARKS	WRITE_SECURE_SETTINGS
ACCESS_WIFI_STATE	SET_ALARM
RESTART_PACKAGES	WRITE_CALL_LOG
WRITE_HISTORY_BOOKMARKS	WRITE_CONTACTS
SET_PREFERRED_APPLICATIONS	RECEIVE_MMS
CHANGE_WIFI_STATE	RECEIVE_WAP_PUSH
BROADCAST_PACKAGE_REMOVED	CAMERA
EXPAND_STATUS_BAR	UPDATE_DEVICE_STATS
DISABLE_KEYGUARD	SET_WALLPAPER
READ_FRAME_BUFFER	PROCESS_OUTGOING_CALLS
CHANGE_WIFI_MULTICAST_STATE	DELETE_CACHE_FILES
CLEAR_APP_CACHE	ACCESS_LOCATION_EXTRA_COMMANDS
SYSTEM_ALERT_WINDOW	GET_PACKAGE_SIZE
ADD_SYSTEM_SERVICE	BLUETOOTH_ADMIN
CHANGE_NETWORK_STATE	BLUETOOTH
READ_CONTACTS	MODIFY_PHONE_STATE

Table 3.7: Detection Rates of SigPID and Anti-Virus Scanners

Method	Detection Rate
SigPID with PART	96.54
SigPID with SVMs	87.67
Mutual Information	82.08
Drebin	93.9
AV1	96.41
AV2	93.71
AV3	84.66
AV4	84.54
AV5	78.38
AV6	64.16
AV7	48.5
AV8	48.34
AV9	9.84
AV10	3.99

Chapter 4

Related Work

In this chapter, we discuss related research efforts in malware detection. While there are many efforts that use static and dynamic analyses to detect malware, they are not closely related to this work, which employs machine learning/data mining to perform malware detection. As such, this chapter only focuses on malware detection efforts based on machine learning and data mining.

Static analysis provides an efficient way to detect the malware targeting Android devices. Enck et al. [14] provided a security service called Kirin for Android. This service certified the applications to mitigate the threats from malware at install time. Kirin performed a security analysis to produce a set of rules to characterize malware and these security rules were used to match undesirable security configurations for applications. They found 5 dangerous applications among 311 most popular applications from Google Play. The results show that Android security configuration is helpful to detect malware.

Felt et al. [6] built a static tool called Stowaway, which can be used to detect overprivilege in Android applications. Stowaway can find what API the application uses and built a map between each API and the permissions this API needs. It covers 85% of Android APIs and this permission map can help to find overprivilege in applications. Overprivilege was found in one third of the 940 applications. The reason is that programmers sometimes follow less privilege strategy and Google dose not provide enough documentation for permissions.

SCanDroid [15] is a static analysis tool to detect the violations of information flow, and it recommends that whether an application can be installed without violating other applications' permissions. However, source codes of the applications are needed to do the analysis, which is the weakness of this tool. CHEX [16] is a static analysis tool to discover permission leakage in Android applications. It can check several types of vulnerabilities, however, the manifest that is related to the Activity's exported attribute is not checked by it. Kantola [17] modifies the configuration semantics to mark Activity classes as public and uses the heuristic-based approach to fix vulnerabilities.

Grace et al. [5] implemented a system, RiskRanker, which can discover dangerous behaviors of an app via static analysis. This system reported 3,281 risky apps, among these apps, 718 are malware samples and 322 of them are zero-day.

DREBIN [9] is a static analysis tool which combines permissions and APIs with machine learning to detect malware. They embedded features in a vector space, dicovered patterns of malware from the vector space, and used these patterns to build the machine learning detection system. Their evaluation results indicate that their proposed work can achieve high detection accuracy. However, their analysis is performed on the devices and therefore, it requires that those devices be rooted.

Dynamic analysis approaches, on the other hand, monitor runtime behaviors. They exercise targeted applications, monitor app activities and collect the relevant data to help with analysis of runtime behaviors.

TaintDroid [7] is a dynamic taint tracking system for tracking multiple sensitive data source simultaneously. TaintDroid provides real time analysis by using Android's virtualized execution environment. 30 popular third-parity applications were monitored using this tool and 68 misuses of private information were found among 20 applications. TaintDroid can be used to monitor sensitive data in order to uncover misbehaving applications.

Zhang et al. presented a framework called VetDroid [18], which is a dynamic analysis platform to reconstruct sensitive behaviors according to the use of permissions. These behaviors can show, i.e., how applications access resources, how these resources are utilized by applications, and analysts can check the sensitive information based on these behaviors. They applied VetDroid to construct malicious behaviors of malicious apps in order to ease malware analysis. It can also find more information leaks than TaintDroid[3]

Zhou et al. [19] proposed a system called DroidRanger, which is a dynamic permission-based behavioral scheme to detect new malware samples from existing families. They identified certain inherent malicious behaviors by applying an heuristics-based strategy. DroidRanger can detect 211 infected applications among 204,040 apps, and they discovered two zero-day malware.

Yan et al. [20] presented a platform called DroidScope, which reconstructs the semantics and exports APIs for three levels of Android device: hardware, OS and Dalvik Virtual Machine. Several analysis tools were also provided to gather instruction traces, API activities and to track information leakage. DroidScope is an effective platform for analyzing malware samples.

AppsPlayground [21] is a framework that automatically analyzes Android applications. It integrates different detection and exploration techniques for the analysis. The evaluation indicates that AppsPlayground can detect privacy leakages and malware effectively.

Machine learning and data mining techniques such as SVM, Random Forest, Decision Tree and clustering algorithms, have been used to classify unknown malware samples into existing families. Researchers have been combining program analysis with machine learning and data mining techniques in order to improve the performance of their detection systems. Although there are many malware detection system based on the signature, applying machine learning and data mining to detect malware has recently gained popularity including detecting malware in Microsoft Windows workstations [22].

Previous work has used Android permissions to detect malware. Huang et al. [23] explored the possibilities of detecting malicious applications based on permissions using machine learning. They retrieved not only all the permissions, but also several easy-to-retrieve features from each APK to help detect malware. Four common machine-learning algorithms, AdaBoost, Naïve Bayes, Decision Tree and SVM, are employed in their system to evaluate the performance. Experimental results show that more than 80% of the malicious samples can be detected by the system. Because the precision is not high, their system can only be used as a first level filter to help detecting malware. A second pass of analysis is still needed for their system. We can achieve a much higher detection rate compared to this work.

DREBIN [9] combines static analysis of permissions and APIs with machine learning to detect malware. They embedded features in a vector space, discovered patterns of malware from the vector space, and used these patterns to build the machine learning detection system. Their evaluation results indicate that their proposed work can achieve high detection accuracy. However, their analysis is performed on the devices, and therefore requires that those devices be rooted. They extracted as many features as possible to help improve performance. However, our work only employs the significant permission features, which reduces the overhead of computation while retaining a satisfying result.

Wang et al. [10] explore the permission-induced risk in Android apps using data mining. They perform an analysis of individual permission and collaborative permissions and apply three ranking methods on the permission features. After the ranking step, they identify risky permission subsets using *Sequential Forward Selection* (*SFS*) and *Principal Component Analysis* (*PCA*). They evaluate their approach using SVM, decision tree and random forest. The result shows that their strategy for identifying risky permissions can achieve a 94.62% detection rate with a 0.6% false positive rate. Our work needs less permissions compared to this work, but a high detection rate can still be achieved.

Schultz et al. [24] use data mining to help with malware detection. They extract static features from executable binaries and use Naive Bayers to find malware patterns. Their unsupervised environment makes the validation of the results more difficult than our supervised approach. Many of these line of work extract features from dynamic behaviors, which needs more computation and run time information. Nevertheless, our permission strategy is not only a static analysis based approach, but we also further reduce the number of permissions needed to make the system more efficient.

FIRMA [25] is a tool that clusters unlabeled malware samples according to network traces. It produces network signatures for each malware family for detection. Work by Perdisci et al. [26] introduces a behavioral malware clustering system based on HTTP traffic. They define a set of similarity metrics of HTTP traces. Their approach then clusters these traces and generates high-quality malware signatures. Nari et al. [27] proposes a framework to automatically classify malware into the existing families. They extract the network behaviors from PCAP files and build a graph based on network activities and flows. They use the characteristics of this graph such as vertex out-degree to be features to classify malware. SMASH [28] make use of HTTP communication to detect attacking campaigns and malicious communication campaigns via a unsupervised data mining approach. Again, these approaches only utilize a subset of information available in the HTTP header.

Chapter 5

Conclusion

In this thesis, we have shown that it is possible to reduce the number of permissions to be analyzed in mobile malware detection, while maintaining high effectiveness and accuracy. SigPID has been designed to extract only significant permissions through a systematic, 3-level pruning approach. Based on our dataset, which includes over 1000 malware, we only need to consider 34 out of 135 permissions to improve the runtime performance by 85.6% while achieving over 90% detection accuracy. The extracted significant permissions can also be used by other commonly used supervised learning algorithms to yield the F-measure of at least 85% in 55 out of 67 tested algorithms. SigPID is highly effective, when compared to the state of the art malware detection approaches as well as existing virus scanner. It can detect 96.54% of malware in the dataset, while other approaches can only detect 3.99% to 96.41%.

Chapter 6

Future Work

In this thesis, we introduce SIGPID, a new system to detect malware based on significant permissions. The evaluation results show that it can work well, with the malware detection rate of 96% when we use only 34 out of 135 permissions. However, supervised learning with permissions alone can not solve all major problems in mobile security. Thus, we need to find other meaningful features that can be used to detect other types of security vulnerabilities and attacks as part of our future research.

In DREBIN [9], besides permissions, they also collect APIs, network address which is helpful to detect the malicious apps. After collecting more features, we believe SIGPID can find other significant patterns from the whole dataset, and build a more effective malware detection detection system that can be used to detect other forms of attacks such as collusion.

In this thesis, we also implemented and evaluated 67 supervised learning algorithms. Different machine learning algorithms are suitable for different datasets. I hope that we can develop a new machine learning algorithm which is more suitable for classifying malicious apps in smart-mobile devices. This will undoubtedly require more insights into the behaviors of such malware, profound thinking, and study into the dataset's type and structure.

While observing the use of permissions and APIs allow us to identify their usage frequencies within an app and across a collection of apps, we can still see that there are some apps that are more sophisticated to simply use this simple observation. Instead, we may need to include both static program structure information (e.g., static call graphs and calling context) and runtime information (e.g., dynamic call graphs) to further classify behavior and pinpoint locations of malicious code. For example, a call to method M3 by method M0 may not instigate any malicious behavior. However, a call to M3 by M1 may instigate malicious behavior due to the arguments passed into M1 by M1. This additional information can help us to classify the apps when they use same permissions or APIs.

One important trend that has been observed by many security analysts and researchers is that malware writers tend to use variations of existing known malware. This provides us with an opportunity to extract code signature (e.g., structure, calling context, variable usage) as a way to detect variations of the same malware. This can increase effectiveness of malware detection in the future.

Bibliography

[1] I. IDC Research, “Smartphone os market share, 2015 q2,” in *IDC Research Report*, 2015.

[2] G. Kelly, “Report: 97% of mobile malware is on android. this is the easy way you stay safe,” in *Forbes Tech*, 2014.

[3] G. DATA, “At a glance,” in *GăDATA MOBILE MALWARE REPORT*, 2015.

[4] Symantec, “Latest intelligence for march 2016,” in *Symantec Official Blog*, 2016.

[5] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, “Riskranker: scalable and accurate zero-day android malware detection,” in *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM, 2012, pp. 281–294.

[6] 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*. ACM, 2011, pp. 627–638.

[7] W. Enck, P. Gilbert, S. Han, V. Tendulkar, 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,” *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 2, p. 5, 2014.

[8] T. Petsas, G. Voyatzis, E. Athanasopoulos, M. Polychronakis, and S. Ioannidis, "Rage against the virtual machine: hindering dynamic analysis of android malware," in *Proceedings of the Seventh European Workshop on System Security*. ACM, 2014, p. 5.

[9] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket," in *Proceedings of the Annual Symposium on Network and Distributed System Security (NDSS)*, 2014.

[10] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang, "Exploring permission-induced risk in android applications for malicious application detection," *Information Forensics and Security, IEEE Transactions on*, vol. 9, no. 11, pp. 1869–1882, 2014.

[11] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *Proc. 20th int. conf. very large data bases, VLDB*, vol. 1215, 1994, pp. 487–499.

[12] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[13] E. Frank and I. H. Witten, "Generating accurate rule sets without global optimization," in *ICML*, vol. 98, 1998, pp. 144–151.

[14] 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*. ACM, 2009, pp. 235–245.

[15] A. P. Fuchs, A. Chaudhuri, and J. S. Foster, “Scandroid: Automated security certification of android,” 2009.

[16] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang, “Chex: statically vetting android apps for component hijacking vulnerabilities,” in *Proceedings of the 2012 ACM conference on Computer and communications security.* ACM, 2012, pp. 229–240.

[17] D. Kantola, E. Chin, W. He, and D. Wagner, “Reducing attack surfaces for intra-application communication in android,” in *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices.* ACM, 2012, pp. 69–80.

[18] Y. Zhang, M. Yang, B. Xu, Z. Yang, G. Gu, P. Ning, X. S. Wang, and B. Zang, “Vetting undesirable behaviors in android apps with permission use analysis,” in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security.* ACM, 2013, pp. 611–622.

[19] 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 *NDSS*, 2012.

[20] L. K. Yan and H. Yin, “Droidscope: seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis,” in *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, 2012, pp. 569–584.

[21] V. Rastogi, Y. Chen, and W. Enck, “Appsplayground: automatic security analysis of smartphone applications,” in *Proceedings of the third ACM conference on Data and application security and privacy.* ACM, 2013, pp. 209–220.

[22] M. Siddiqui, M. C. Wang, and J. Lee, “A survey of data mining techniques for malware detection using file features,” in *Proceedings of the 46th annual southeast regional conference on xx.* ACM, 2008, pp. 509–510.

[23] C.-Y. Huang, Y.-T. Tsai, and C.-H. Hsu, “Performance evaluation on permission-based detection for android malware,” in *Advances in Intelligent Systems and Applications-Volume 2.* Springer, 2013, pp. 111–120.

[24] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo, “Data mining methods for detection of new malicious executables,” in *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on.* IEEE, 2001, pp. 38–49.

[25] M. Z. Rafique and J. Caballero, “Firma: Malware clustering and network signature generation with mixed network behaviors,” in *Research in Attacks, Intrusions, and Defenses.* Springer, 2013, pp. 144–163.

[26] R. Perdisci, W. Lee, and N. Feamster, “Behavioral clustering of http-based malware and signature generation using malicious network traces.” in *NSDI*, 2010, pp. 391–404.

[27] S. Nari and A. A. Ghorbani, “Automated malware classification based on network behavior,” in *Computing, Networking and Communications (ICNC), 2013 International Conference on.* IEEE, 2013, pp. 642–647.

[28] J. Zhang, S. Saha, G. Gu, S.-J. Lee, and M. Mellia, “Systematic mining of associated server herds for malware campaign discovery,” in *Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on.* IEEE, 2015, pp. 630–641.