Significant Permission Identification for Machine Learning Based Android Malware Detection

Jin Li*[†], Lichao Sun*[‖], Qiben Yan[‡]**, Zhiqiang Li[‡], Witawas Srisa-an[‡] and Heng Ye[§][†]
[†]School of Computer Science and Educational Software, Guangzhou University, Guangdong, China
[‖]Department of Computer Science, University of Illinois at Chicago, Chicago, IL, USA
[‡]Department of Computer Science and EngineeringUniversity of Nebraska–Lincoln, Lincoln, NE, USA
[§]Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University, 3 Shangyuancun, Beijing, China
* Both authors contributed equally

***Abstract*—The alarming growth rate of malicious apps has become a serious issue that sets back the prosperous mobile ecosystem. A recent report indicates that a new malicious app for Android is introduced every 10 seconds. To combat this serious malware campaign, we need a scalable malware detection approach that can effectively and efficiently identify malware apps. Numerous malware detection tools have been developed, including system-level and network-level approaches. However, scaling the detection for a large bundle of apps remains a challenging task. In this paper, we introduce SIGPID, a malware detection system based on permission usage analysis to cope with the rapid increase in the number of Android malware. Instead of extracting and analyzing all Android permissions, we develop 3-levels of pruning by mining the permission data to identify the most significant permissions that can be effective in distinguishing between benign and malicious apps. SIGPID then utilizes machine-learning based classification methods to classify different families of malware and benign apps. Our evaluation finds that only 22 permissions are significant. We then compare the performance of our approach, using only 22 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 while incurring the analysis times that are 4 to 32 times less than those of using all permissions. Compared against other state-of-the-art approaches, SIGPID is more effective by detecting 93.62% of malware in the data set, and 91.4% unknown/new malware samples.**

I. INTRODUCTION

Android is currently the most used smart-mobile device platform in the world, occupying 85% of market share [1]. As of now, there are nearly 3 million apps available for downloading from Google Play, and more than 65 billion downloads to date [2]. Unfortunately, the popularity of Android also spurs interests from cyber-criminals who create malicious apps that can steal sensitive information and compromise mobile systems. Unlike other competing smart-mobile device platforms, such as iOS, Android allows users to install applications from unverified sources such as third party app stores and file sharing websites. The malware infection issue has been so serious that a recent report indicates that 97% of all mobile malware target Android devices [3]. In 2016 alone, over 3.25 million new malicious Android apps have been uncovered. This roughly translates to an introduction of a new malicious Android app every 10 seconds [4]. These malicious apps are created to perform different types of attacks in the form of trojans, worms, exploits, and viruses. Some notorious malicious apps have more than 50 variants, which makes it extremely challenging to detect them all [5].

To address these elevating security concerns, researchers and analysts have used various approaches to develop Android malware detection tools [6]–[19]. For example, RISKRANKER [6] utilizes static analysis to discover malicious behaviors in Android apps. However, static analysis approaches generally assume more behaviors are possible than actually would be, which may lead to a large number of false positives. To improve the analysis accuracy, researchers have also proposed various dynamic analysis methods to capture real-time execution context. For example, TAINTDROID [8] dynamically tracks multiple sensitive data source simultaneously using tainting analysis. However, dynamic analysis approaches, in general, need adequate input suites to sufficiently exercise execution paths. As we can use cloud computing to satisfy those requirements, the privacy concerns then become potential problem. To protect data's privacy, Jin et al. [20] [21] [22] proposed methods to support ciphertext operation. Also, Petra et al. [23], [24] show that there is another broad range of anti-analysis techniques can be employed by advanced malware to successfully evade dynamic analysis based malware detection.

Recently, more efforts have been spent on analyzing apps' behavioral data both in the Android system and other online data process system [25]–[27]. The apps' requested permissions have been used to enforce least-privilege, which to some extent indicate the apps' functionalities as well as runtime behaviors. As a result, researchers use machine learning and data mining techniques to detect Android malware based on permission usage. 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 incorporating as many features as possible to aid detection. However, using more features leads to increased modeling complexity, which increases the computational overhead of their system. Considering the

**Corresponding author: yan@unl.edu

large amount of new malicious apps, we need a detection system that can operate efficiently to identify these apps. Google also identifies 24 permissions out of the total of more than 300 permissions as "dangerous" [28]. At first glance, the list of dangerous permissions can be used as a guideline to help identify malicious applications. Yet, as will be shown in this paper, using just this list to identify malware still yields suboptimal detection effectiveness.

In this paper, 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, machine-learning based classification algorithms are used to classify different types of malware and benign apps.

The results of our empirical evaluation show that SIGPID can drastically reduce the number of permissions that we need to analyze to just 22 out of 135 (84% reduction), while maintaining over 90% malware detection accuracy and F-measure when Support Vector Machine (SVM) is used as the classifier. We also find that the number of significant permissions identified by our approach (22) is lower than the number of "dangerous" permissions identified by Google (24). Moreover, only 8 permissions jointly appear on our list and their list. This is because, as a data-driven approach, SIGPID dynamically determines significant permissions based on actual usage by the applications instead of statically defining dangerous permissions based on their intended services. This fundamental difference allows our approach to detect more malware than the approach that uses the dangerous list alone. 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 all these algorithms. Furthermore, we compare the accuracy and running-time performance of our approach against two state-of-the-art approaches, DREBIN [9], PERMISSION-INDUCED RISK MALWARE DETECTION [29], and existing virus scanners. Again, we find that our approach can detect more malware samples than the other approaches with significantly less overhead.

In summary, our paper makes the following contributions:

1) We develop SIGPID, an approach that identifies an essential subset of permissions (significant permissions) that can be used to effectively identify Android malware. By using our technique, the number of permissions that needs to be analyzed is reduced by 84%.
2) We evaluate the effectiveness of our approach using only a fifth 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 as well as just the dangerous permission list. Compare with other state-of-the-art malware detection approaches, we find that SIGPID is more effective by detecting 93.62% of malicious apps in the data set and 91.4% unknown malware.
3) To show that the approach can work generically with a wide range of supervised learning algorithms, we apply SIGPID with 67 commonly used 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.

The rest of this paper is organized as follows. Section II illustrates the design details of the proposed SIGPID. Section III reports the results of our empirical evaluations. Section IV compares our work to other related work. The last section concludes this paper.

II. INTRODUCING SIGPID

The goal of *Significant Permission IDentification* (SIGPID) system is to achieve high malware detection accuracy and efficiency while analyzing the minimal number of permissions. To achieve this goal, our system extracts permission uses from application packages, but instead of focusing on all the requested permissions, SIGPID mainly focuses on permissions that can reliably improve the malware detection rate. This approach, 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 to reduce analysis efforts. Our system consists of three major components, designed based on real-time data analysis: (i) permission ranking with negative rate; (ii) support based permission ranking; and (iii) permission mining with association rules. 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 1. We then describe the key components in the remainder of this section.

A. Multi-Level Data Pruning (MLDP)

The first component of SIGPID is the multi-level data pruning process to identify significant permissions to eliminate the need of considering all available permissions in Android. No app requests all the permissions, and the ones that an app requests are listed in the Android application package (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.

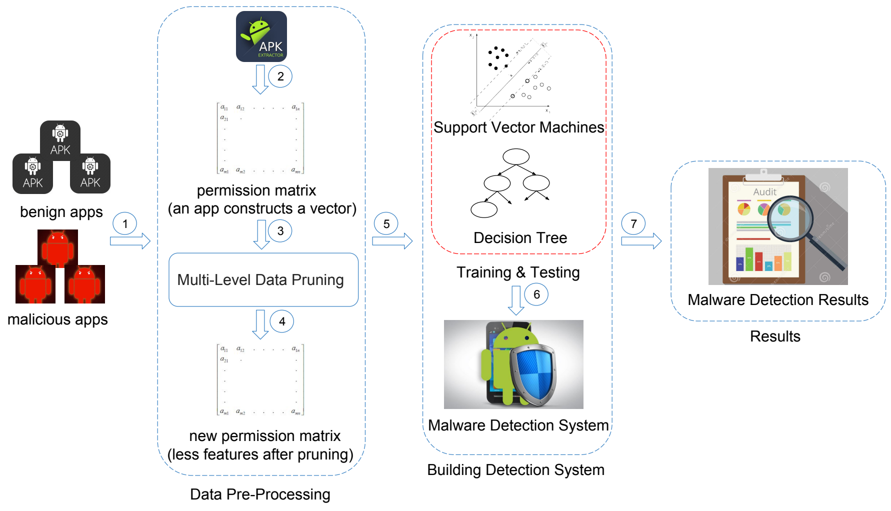

Fig. 1: Architectural Overview of SigPID

We propose three levels of data pruning methods to filter out permissions that contribute little to the malware detection effectiveness. Thus, they can be safely removed without negatively affecting malware detection accuracy. The complete three-step procedure is illustrated in Figure 2. We then describe each level in the pruning process.

1) Permission Ranking with Negative Rate (PRNR): 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 [9] [29], and we do not need to analyze all the permissions to build an effective malware detection system.

As a result, on one hand, our focus is more on the permissions that create high risk attack surfaces and are frequently requested by malware samples. On the other hand, the permissions that are rarely requested by malware samples are also good indicators in differentiating between malicious and benign apps. Therefore, Our pruning procedure identifies both types of highly differentiable permissions so that we can use this information to classify malicious and benign apps. At the same time, we 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. Therefore, our approach prunes permission INTERNET.

To identify these two types of significant permissions, we design a permission ranking scheme to rank permissions based on how they are used by malicious and benign apps. Ranking is not a new concept. Prior works have also used generic permission ranking strategy such as mutual information to identify high risk permissions [29]. However, their approaches tend to only focus on high-risk permissions, and ignore all the low-risk permissions, which are defined as significant permissions in our approach. The reason that prior works ignoring low-risk permissions is that they are interested in identifying the permissions abused by malware, while our goal is to differentiate between malware and benign apps. In essence, risky permissions only focus on the permissions which can help detect the malware, while significant permissions not only care about the identification of the malware, but also take into account whether benign apps can be identified or not.

Our approach, referred to as *Permission Ranking with Negative Rate* or PRNR, provides a concise ranking and comprehensible result. 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 the support of permissions from matrices M and B, we first check their sizes. Typically, the number of benign apps (310,926 in this paper) tends to be much larger than the number of malicious apps (5,494 samples in this paper); therefore, the size of B is much larger than the size of M. With our ranking scheme, we prefer the data set on the two matrices to be balanced. Training over imbalanced dataset can lead to skewed models [30]. To balance the two matrices, we use Equation 1 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. In case that the number of rows of B is bigger than that of M, we have:

$$S_B(P_j) = \frac{\sum_i B_{ij}}{size(B_j)} * size(M_j), \tag{1}$$

where P_j denotes the j^{th} permission, and $S_B(P_j)$ represents the support of j^{th} permission in matrix B. PRNR can then be implemented using Equation 2:

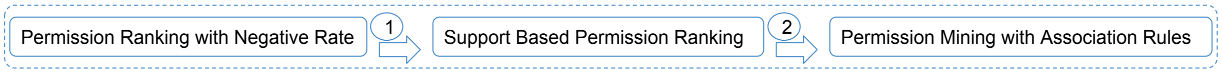
Permission Ranking with Negative Rate
1
Support Based Permission Ranking
2
Permission Mining with Association Rules

Fig. 2: Multi-Level Data Pruning

$$R(P_j) = \frac{\sum_i M_{ij} - S_B(P_j)}{\sum_i M_{ij} + S_B(P_j)} \tag{2}$$

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$_j$) = 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. If R(P$_j$) = 0, this means that P$_j$ has very little impact on malware detection effectiveness. Since both -1 and 1 are important, we build two ranked lists: one list is generated in an ascending order based on the value of R(P$_j$), and the other list is generated in a descending order.

Next, we design a novel *Permission Incremental System* (*PIS*) to 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 system. We then evaluate malware detection using the following metrics: true positive rate (TPR), false positive rate (FPR), precision($P = \frac{\text{TPR}}{\text{TPR+FPR}}$), recall ($R = \frac{\text{TPR}}{\text{TPR+FNR}}$), accuracy, and F-measure($\frac{\text{2P*R}}{\text{P+R}}$). Then, we choose the top three permissions in both lists to build malware detection system. We repeat the process again while increasing the number of top permissions to use for malware detection until the detection metrics plateau. 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. Within our evaluation dataset with $5,494$ benign apps from $310,926$, the total number of distinct permissions requested by all the apps is 135. After applying PRNR, we find that using 95 permissions performs nearly as well as using the complete 135 permissions. Detailed results of our evaluation is provided in Section III.

2) Support Based Permission Ranking (SPR): 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 performance. For instance, we find the permission BIND_TEXT_SERVICE only in benign apps. As a result, we may consider that any app that uses BIND_TEXT_SERVICE is benign. However, this permissions is in fact only used by one app out of over 310,926 benign apps. Consequently, only relying on the ranked rate provided by PRNR might be inaccurate. By incorporating permission support, we can rule out permissions with low support to further improve the malware classification accuracy.

Similar to PRNR, we use PIS to find the least number of permissions with high support while yielding good accuracy. Within our dataset, after applying SPR, we are able to reduce the number of significant permissions to just 25 or 19% of the total permissions. Again, the evaluation results are presented in Section III.

3) Permission Mining with Association Rules (PMAR): After pruning 110 of 135 permissions by using PRNR and SPR with PIS, we want to further explore approaches that can remove non-influential permissions even more. By inspecting the reduced permission list that contains 25 significant permissions, we find three pairs of permissions that always appear together in an app. For example, permission WRITE_SMS and permission READ_SMS are always used together. They also both belong to the Google's "dangerous" permission list. Yet, it is unnecessary to consider both permissions, as one of them is sufficient to characterize certain behaviors. As a result, we can associate one, which has a higher support, to its partner. In this example, we can remove permission WRITE_SMS. In order to find permissions that occur together, we propose a permission mining with association rules (PMAR) mechanism using association rule mining algorithm [31].

Association rule mining has been used for discovering meaningful relations between variables in large databases. For instance, if event A and B always co-occur, it is highly likely that these two events are associated. In this paper, we only consider rules with high confidence, so that applying PMAR will only produce a small number of rules. We employ *Apriori*, a commonly used association mining algorithm, to generate the association rules. Apriori [31] 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. Here, we only want to generate the association rules with high confidence even if the permissions have small support values.

In the end, we identify 3 association rules based on our permission dataset, corresponding to three pairs of associated permissions, when we set 96.5% minimum confidence and 10% minimum support. Finally, we are able to remove three additional permissions, leaving 22 permissions that we consider as significant.

B. Machine-Learning based Malware Detection using Significant Permissions

We first use SVM and a small dataset to test our proposed MLDP model. SVM determines a hyperplane that separates both classes with a maximal margin based on the training dataset that includes benign and malicious applications. 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 are 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 identified 135 permissions (baseline) and malware detection using MLDP for each supervised machine learning algorithm, as shown in Section III-C. We observe that, by analyzing all permissions, machine learning algorithms with a tree structure, usually build better malware detection compared to others. Among all the machine learning algorithms with a tree structure, our method works best on decision tree, which is a very common classification method in machine learning. Decision tree should use training dataset to build a classifier to decide which class the testing data should be, requires a lot of pre-processing work before the classifier was built. When there are too many attributes of training dataset, decision tree hits a poor accuracy and the training phase tend to take more time and memory. So, the pruning of the decision tree is imperative, which is consistent with our MLDP.

Consequently, it is more advantageous to use MLDP to perform malware detection as it can be as effective while notably conserving time and memory. Since the time and memory are limited in common computers, we can outsourcing the task to cloud in a suitable way to boost efficiency. Because of the permissions can be recorded perfectly by using binary vector, we could remove the mapping table and doing some permutation before the outsourcing to preserve data's privacy. We will provide more details in the next section.

III. Evaluation

In this section, we evaluate the malware detection effectiveness of the SigPID system. We first use SVM algorithm to evaluate the classification performance of our proposed MLDP model. Through pruning, our system, as previously mentioned, identifies only 22 significant permissions (a reduction of 84%). On the other hand, Google lists 24 Android permissions as dangerous. We also use these 24 permissions to build a malware detection system as part of our empirical evaluation (referred to as "Android" method hereinafter). As mentioned earlier, MLDP consists of three main components: permission ranking with negative rate, support based permission ranking, and permission mining with association rules. We evaluate the malware detection performance by enabling these multiple levels sequentially to verify the performance improvement contributed by each level of permission mining procedure. In addition, we also evaluate the runtime efficiency of multi-level data pruning.

In most experiments, SVM algorithm is employed to perform malware detection. However, we also illustrate generality of SigPID by employing 67 other commonly used machine learning algorithms. We then identify the five best performing algorithms and employ them to help detect malware in our second collection of apps. Finally, we compare the classification effectiveness of SigPID with results of approaches using other state-of-the-art permission ranking methods such as Mutual Information [29] as well as signature based malware detection commonly employed by commercial virus scanners.

A. Collected Data Set

We generate ten datasets by randomly choosing 5,494 benign apps from 310,926 benign apps, downloaded from Google play store in June 2013 [9], to carry out cross-validation. The malicious apps are classified into 178 families, and the benign apps are grouped into a single family. We select the number of benign apps to be the same as malicious apps to maintain balance during training, as the imbalanced dataset can result in skewed models [30]. We also create another newer collection of apps in June 2015 containing 2,650 new malware samples with no overlap with the initial sets. This new collection is used to evaluate the effectiveness of our detection system to pinpoint newer unknown malware. As the amount of benign apps is much larger than the malicious one, it remains uncertain whether our algorithm can be applied to larger datasets. So we performed another experiment over a much large dataset containing 54,694 different malware samples download from both Google Play and Anzhi Store from Jan 2012 to Dec 2014 [29]. We compared our experiments using three different datasets, which can be seen in Table .V. 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.

B. Evaluating Effectiveness of MLDP

We report the effectiveness of each component of multi-level data pruning as well as the effectiveness of the approach of simply using dangerous permissions provided by Google. The results are presented in Table I. As shown, the simple approach of using dangerous permission to detect malware (last row) yields about the same recall rate or True Positive Rate (TPR) as that of PRNR using 95 permissions (column 4). When we reduce the numbers of permissions to 25 and 22, using SPR and PMAR, respectively, we achieve higher recall rates of over 91%. However, this higher malware detection effectiveness comes at a cost of higher False Positive Rate (FPR) of over 8.5% (column 5).

TABLE I: Detection Results using SVM with Multi-Level Data Pruning and Android Dangerous Permissions

# of Permissions	Approach	Precision (%)	Recall (TPR, %)	FPR (%)	FM (%)	ACC (%)
95	PRNR	96.39	85.78	3.22	90.77	91.28
25	PRNR+SPR	90.64	91.77	9.56	91.17	91.10
22	PRNR+SPR+PMAR	91.55	91.22	8.54	91.34	91.34
135	All Permissions	98.81	83.73	1.01	90.65	91.36
24	Dangerous Permissions	98.64	85.12	1.12	91.38	91.97

Discussion. To further explain the process of determining the number of significant permissions after each pruning technique is applied, we present the performance changing curve after using PRNR w.r.t. the increasing number of permissions in Fig. 3, which shows the key performance metrics of using PRNR with PIS as we gradually vary the number of permissions. We also present performance changing curve after using SPR in Fig. 4, which shows the deviation of key performance metrics after the application of SPR.

According to Fig. 3 and Fig. 4, 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 90%. One keen observation is that the recalls, accuracies and F-measures plateau after the number of permissions reaches 90 in Fig. 3, and the permissions reaches 10 in Fig. 4. Maximum recall is achieved when we use 25 permissions.

Next, we implement permission mining with association rules (PMAR) to perform the last layer of permission mining. Here, we find 3 rules satisfying our associative requirements when we set our confidence level to 96.5%. The association rules mining shows that permission WRITE_SMS and permission READ_SMS have a 98.4034% chance to appear together. Meanwhile, in the cases when they do not appear together, 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, the applications requesting permission READ_SMS will be likely to be classified as malware. In addition, we find that WRITE_HISTORY_BOOKMARKS and READ_HISTORY_BOOKMARKS, WAKE_LOCK and READ_PHONE_STATE have a high chance of being cooccured/associated. After pruning these 3 more permission features in the dataset, we only retain 22 features. We then observe that higher precision is achieved with the new model employing 22 instead of 25 permissions in Table I.

TABLE II: Rankings by PRNR and Mutual Information

MLDP	
22 Permissions	
ACCESS_WIFI_STATE	READ_LOGS
CAMERA	READ_PHONE_STATE
CHANGE_NETWORK_STATE	READ_SMS
CHANGE_WIFI_STATE	RECEIVE_BOOT_COMPLETED
DISABLE_KEYGUARD	RESTART_PACKAGES
GET_TASKS	SEND_SMS
INSTALL_PACKAGES	SET_WALLPAPER
READ_CALL_LOG	SYSTEM_ALERT_WINDOW
READ_CONTACTS	WRITE_APN_SETTINGS
READ_EXTERNAL_STORAGE	WRITE_CONTACTS
READ_HISTORY_BOOKMARKS	WRITE_SETTINGS
Mutual Information	
Top 22 Permissions	
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

We then compare the list of significant permissions generated by our approach to the top 22 permissions identified by another permission ranking method called Mutual Information [29]. It uses 40 permissions and achieves 86.4% TPR (recall) and 1.4% FPR. We list these permissions in Table II.

We conclude that different permission ranking methods induce different ranking lists. For example, 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. When we compare our list of 22 significant permissions to the list of 24 dangerous permissions identified by Google, we notice that there are only 8 permissions that appear on both lists. Therefore, we believe our algorithm can retain more significant permissions by pruning less important or meaningless permissions compared with other permission ranking methods. This allows our approach to identify more malicious apps (with higher recall rate), which is an important property of an effective malware detector.

Note that the pruning order of MLDP can be rearranged. Based on our results, we find that SPR may prune more permissions than PRNR if we switch them. As a result, we can also use an alternative pruning order such as SPR, PRNR, and then PMAR.

C. Evaluating Generality of MLDP

In order to show the generality of MLDP, we experiment with using different malware detection models based on 67 machine learning algorithms in Weka [32]. We experiment 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. Fig. 5 reports the top 5 malware detection models that achieve above 96% F-measure, and above 94% in other performance measures, thereby proving the detection model can be generalized to classify the malware and benign apps. Table III report the best machine learning algorithms for our proposed approach (Functional Tree or FT in this case) and the approach of using 24 Android dangerous permissions (Random Forest). We select the algorithm that yields the highest recall for each of these two approaches. As shown, even with the most optimal algorithms, our approach still yields 3.27% higher recall rate than using the dangerous permission list. However, the false positive rate is also 1.09% higher, which is reasonable considering the improvement in other evaluation metrics.

TABLE III: Optimal ML Algorithms For SigPID and Android Dangerous Permissions

Num_of_Feature	Best ML	Precision	Recall(TPR)	FPR	FM	ACC
SigPID (22)	FT	97.54%	93.62%	2.36%	95.54%	95.63%
Android (24)	RandomForest	98.61%	90.35%	1.27%	94.30%	94.54%

TABLE IV: Training and Testing Time with Different Numbers of Permissions

Num_of_Feature	22	40		135	
Algorithm	Time(Seconds)	Time	More Time	Time	More Time
RandomCommittee	1.376	2.078	51.02%	7.995	481.03%
RotationForest	47.303	71.887	51.97%	236.944	400.91%
FT	0.731	2.14	192.75%	24.55	3258.41%
PART	16.673	24.645	47.81%	104.74	528.20%
RandomForest	14.028	20.045	42.89%	59.991	327.65%
SVM	2.4722	2.7604	11.66%	3.6773	48.75%

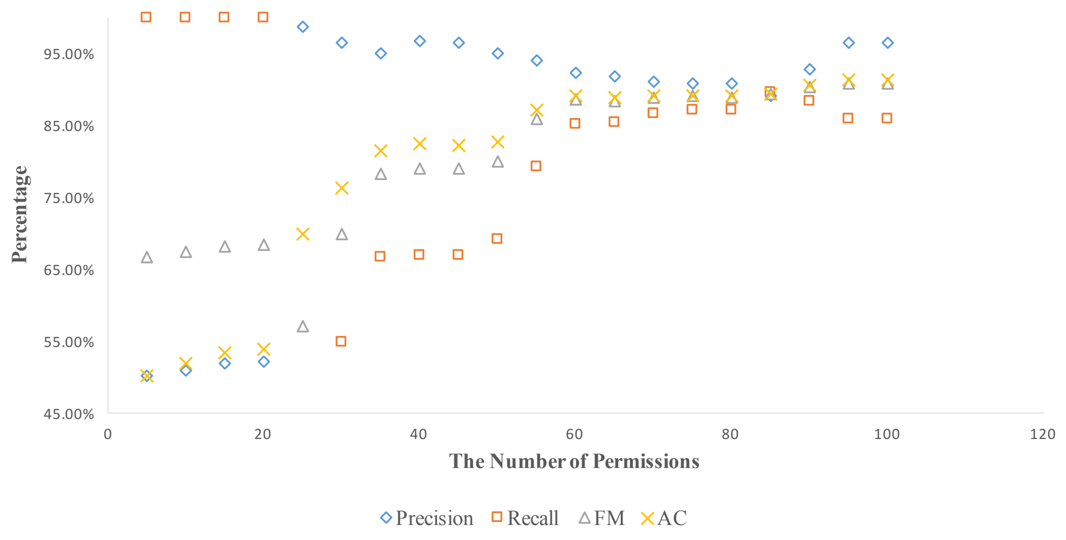

Fig. 3: Malware Classification Performance of Permission Incremental System with PRNR

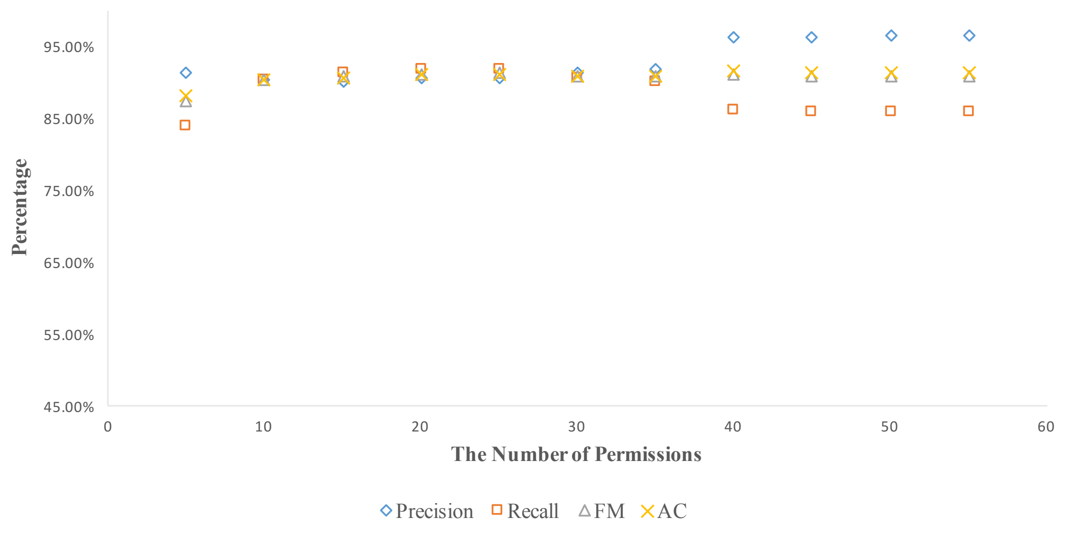

Fig. 4: Malware Classification Performance of Permission Incremental System with SPR

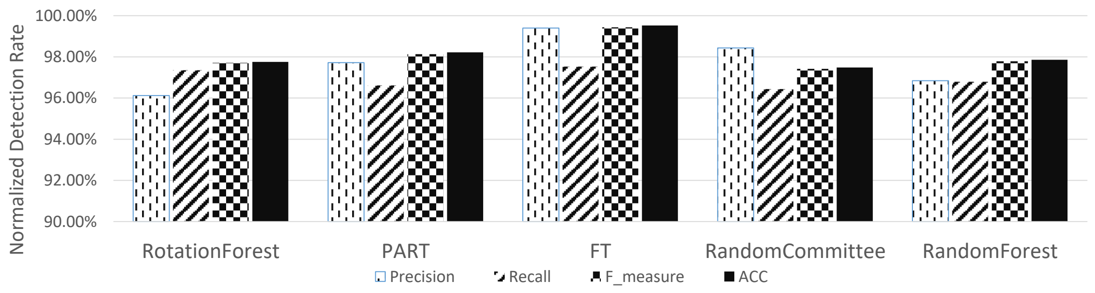

Fig. 5: Results with Top 5 Machine Learning Algorithms

In Table IV, we have the average processing times of the five top performing machine learning algorithms, based on recall rates of using 22, 40 [29], and 135 permissions, respectively. As shown, FT is the most efficient machine learning algorithm. When FT is used with the 22 significant permissions, the processing time is only 0.7 seconds on average, compared to 24.55 seconds from the malware detection model using 135 permissions.

We then apply our approach and malware detection based on the list of dangerous permissions to detect a different collection of apps containing 2,650 real-world malware. Again, we apply the top five machine learning algorithms for this evaluation. We report the result in Figure 6, which shows that our approach consistently detects 4% to 5.5% more malware than the approach that uses 24 dangerous permissions, achieving up to 91.4% detection rate.

Discussion. The proposed malware detection approach incurs anywhere from 4 times (when rotation forest is used) to 32 times (when functional tree is used) less processing times than those incurred by an approach that analyzes the complete permission list. Additionally, smaller feature set can also reduce memory consumption.

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.

When we use our approach and the approach that uses 24 dangerous permissions to detect a new collection of malware, we see a similar pattern, in which our system consistently produces higher recall rates. This experiment provides the evidence that the proposed approach can be used to effectively detect malware beyond the initial data set that we used for training and testing, which shows the potential applicability of SIGPID in practice.

D. Applicability of SIGPID

TABLE V: Testing Results using Different Datasets

Num_of_Malapp	2,650	5,494	54,694
Precision	98.83%	97.54%	95.15%
Recall	94.4%	93.62%	92.17%
FPR	1.17%	2.36%	4.85%
FM	94.97%	95.54%	93.63%
ACC	96.47%	95.63%	93.67%

As described before, we have collected three different real-world datasets which contain 2,650, 5,494 and 54,694 malapps, respectively. After evaluating the applicability of our SIGPID using these datasets, we present the results in Table V, which shows that the FPR increases slowly with the growing size of datasets. However, our method can retain a high detection rate despite the size of the dataset is huge.

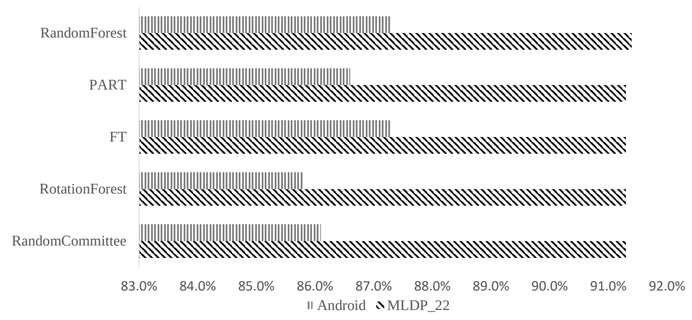

Fig. 6: Detection Performance using Unknown Real-World Malware

E. Comparison with Other Approaches

In this section, we compare our detection results with other state-of-the-art 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. We did not reimplement their approach since it requires significant program analysis in addition to permission analysis. Instead, we compare our results with their reported results.

PERMISSION-INDUCED RISK MALWARE DETECTION [29] 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. We reimplemented their approach for comparison. Note that in their paper, they used a different data set and the ratio of their malicious and benign apps in their dataset is dominated by benign apps. As such, their reported results, especially false positive rate, are significantly different than the results achieved using our data set.

The comparison results are shown in Table VI. 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]. When we combine SIGPID with FT, we can achieve the highest detection rate (93.62%) using only 22 permissions.

Discussion. When we compare the result of our work to other approaches that only consider risky permissions, SIGPID considers a broader criteria that also include non-risky permissions (e.g., READ_CALL_LOG), which are only used in benign apps and have high support values. We deem the risky and non-risky permissions with high support values as *significant permissions*, allowing SIGPID to be more effective in distinguishing between malicious and benign apps than other existing approaches.

We also notice that the permission lists used by DREBIN contain many meaningless features. It is possible that performance improvements can be achieved by integrating SigPID with FT into DREBIN to improve both malware detection accuracy and running time performance. We will explore this integration in our future 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 matching; 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 FT. DREBIN is a more complex malware detection approach that

TABLE VI: Detection Rates of SIGPID and Anti-Virus Scanners

SigPID w/ FT	SigPID w/ SVM	Mutual Information [29]	Drebin [9]	AV1	AV2	AV3	AV4	AV5	AV6	AV7	AV8	AV9	AV10
93.62%	91.22%	86.4%	93.90%	96.41%	93.71%	84.66%	84.54%	78.38%	64.16%	48.50%	48.34%	9.84%	3.99%

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.

IV. Related Work

Previous research efforts have used Android permissions to detect malware. Huang et al. [33] explored the possibilities of detecting malicious applications based on permissions using machine learning mechanisms. 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, Naive Bayes, Decision Tree and SVM [34], 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 detect malware. A second pass of analysis is still needed for their system. Our proposed approach 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. [29] explored 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. Their low false positive rate is brought by their use of a large benign datasets (80% of 310,926 benign apps) for model training, which incurs considerable overhead and produces skewed model. As such, the reimplementation of their method produces different results as shown in Section III-E. Moreover, our work needs less permissions compared to this work, but a high detection rate can still be achieved.

V. Conclusion

In this paper, we have shown that it is possible to reduce the number of permissions to be analyzed for 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 2,000 malware, we only need to consider 22 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 scanners. It can detect 93.62% of malware in the data set, and 91.4% unknown/new malware.

Acknowledgment

We thank the anonymous reviewers for their insightful feedback on our work. This work is supported in part by NSF, DARPA and Maryland Procurement Office under grant numbers CNS-1566388, FA8750-14-2-0053 and H98230-14-C-0140, respectively. Any opinions, findings, conclusions, or recommendations expressed here are those of the authors and do not necessarily reflect the views of the funding agencies or the U.S. Government.

References

[1] IDC, "Smartphone os market share, 2017 q1." [Online]. Available: https://www.idc.com/promo/smartphone-market-share/os

[2] Statista, "Cumulative number of apps downloaded from the google play as of may 2016." [Online]. Available: https://www.statista.com/statistics/281106/number-of-android-app-downloads-from-google-play/

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

[4] G. DATA, "8,400 new android malware samples every day." [Online]. Available: https://www.gdatasoftware.com/blog/2017/04/29712-8-400-new-android-malware-samples-every-day

[5] Symantec, "Latest intelligence for march 2016," in *Symantec Official Blog*, 2016.

[6] 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.

[7] 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.

[8] 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.

[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] C. Yang, Z. Xu, G. Gu, V. Yegneswaran, and P. Porras, "Droidminer: Automated mining and characterization of fine-grained malicious behaviors in android applications," in *European Symposium on Research in Computer Security*. Springer, 2014, pp. 163–182.

[11] M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. Van Der Veen, and C. Platzer, "Andrubis–1,000,000 apps later: A view on current android malware behaviors," in *Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS), 2014 Third International Workshop on*. IEEE, 2014, pp. 3–17.

[12] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck, "Appcontext: Differentiating malicious and benign mobile app behaviors using context," in *Software engineering (ICSE), 2015 IEEE/ACM 37th IEEE international conference on*, vol. 1. IEEE, 2015, pp. 303–313.

[13] S. Wang, Q. Yan, Z. Chen, B. Yang, C. Zhao, and M. Conti, "Textdroid: Semantics-based detection of mobile malware using network flows."

[14] Z. Li, L. Sun, Q. Yan, W. Srisa-an, and Z. Chen, "Droidclassifier: Efficient adaptive mining of application-layer header for classifying android malware," in *International Conference on Security and Privacy in Communication Systems*. Springer, 2016, pp. 597–616.

[15] Z. Chen, Q. Yan, H. Han, S. Wang, L. Peng, L. Wang, and B. Yang, "Machine learning based mobile malware detection using highly imbalanced network traffic," *Information Sciences*, 2017.

[16] S. Wang, Q. Yan, Z. Chen, B. Yang, C. Zhao, and M. Conti, "Detecting android malware leveraging text semantics of network flows," *IEEE Transactions on Information Forensics and Security*, 2017.

[17] J. Z. L. H. P. S. Y. Lichao Sun, Xiaokai Wei and W. Srisa-an, "Contaminant removal for android malware detection systems," in *Proceedings of IEEE International Conference on Big Data*, 2017.

[18] L. Sun, Y. Wang, B. Cao, P. S. Yu, W. Srisa-an, and A. D. Leow, "Sequential keystroke behavioral biometrics for mobile user identification via multi-view deep learning," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD)*, 2017.

[19] S. Wang, Q. Yan, Z. Chen, B. Yang, C. Zhao, and M. Conti, "Textdroid: Semantics-based detection of mobile malware using network flows," in *IEEE INFOCOM 2017 Workshop: MobiSec 2017, Atlanta, GA, USA*, May 2017.

[20] J. Li, Y. Zhang, X. Chen, and Y. Xiang, "Secure attribute-based data sharing for resource-limited users in cloud computing," *Computer and Security*, vol. 72, pp. 1–12, 2018.

[21] P. Li, J. Li, Z. Huang, T. Li, C. Gao, X. Liu, and K. Chen, "Multi-key privacy-preserving deep learning in cloud computing," *Future Generation Computer Systems*, vol. 74, pp. 76–85, 2017.

[22] P. Li, J. Li, Z. Huang, C. Gao, W. Chen, and K. Chen, "Privacy-preserving outsourced classification in cloud computing. cluster computing," *Cluster Computing*, pp. 1–10, 2017.

[23] 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.

[24] X. Cao, L. Liu, W. Shen, A. Laha, J. Tang, and Y. Cheng, "Real-time misbehavior detection and mitigation in cyber-physical systems over wlans," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 1, pp. 186–197, 2017.

[25] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 3, p. 15, 2009.

[26] K. Bu, M. Xu, X. Liu, J. Luo, S. Zhang, and M. Weng, "Deterministic detection of cloning attacks for anonymous rfid systems," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 6, pp. 1255–1266, 2015.

[27] T. Cruz, L. Rosa, J. Proenca, L. A. Maglaras, M. Aubigny, L. Lev, J. Jiang, and P. Simoes, "A cybersecurity detection framework for supervisory control and data acquisition systems," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 6, pp. 2236–2246, 2016.

[28] G. Android, "Requesting permissions." [Online]. Available: https://developer.android.com/guide/topics/permissions/requesting.html

[29] 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.

[30] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, Sept 2009.

[31] 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.

[32] 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.

[33] 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.

[34] A. Jindal, A. Dua, K. Kaur, M. Singh, N. Kumar, and S. Mishra, "Decision tree and svm-based data analytics for theft detection in smart grid," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 3, pp. 1005–1016, 2016.