# SigPID: Significant Permission Identification for Android Malware Detection

Lichao Sun, Zhiqiang Li, Qiben Yan, Witawas Srisa-an and Yu Pan
University of Nebraska–Lincoln
Lincoln, NE 68588
{lsun,zli,qyan,witty,ypan}@cse.unl.edu

## Abstract

*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 paper, 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.* SIGPID *then utilizes classification algorithms to classify different families of malware and benign apps. Our evaluation finds that only 22 out of 135 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 smaller that those of using all 135 permissions. When we compare the detection effectiveness of* SIGPID *to those of other approaches,* SIGPID *can detect 93.62% of malware in the data set, and 91.4% unknown malware.*

## 1 Introduction

Android is currently the most used smart-mobile device platform in the world, occupying 82.8% of market share [10]. 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 [11]. In 2015 alone, over 2.33 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. Some types of malicious apps have more than 50 variants, which makes it extremely challenging to detect them all [13].

To address these elevating security concerns, researchers and analysts have used various approaches to develop malware detection tools [4–6]. For example, RISKRANKER [6] uses static analysis to discover malicious behaviors in Android apps. However, static analysis approaches in general can 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 [4] dynamically tracks multiple sensitive data source simultaneously. However, dynamic analysis approaches, in general, need adequate input suites to sufficiently exercise execution paths. Furthermore, Petra et al. [12] show that a broad range of anti-analysis techniques can be employed by malware to successfully evade dynamic analysis.

Recently, more efforts have been spent on analyzing permission in the Android system. Researchers have used machine learning and data mining techniques to detect Android malware based on permission usage. For example, DREBIN [2] 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. Considering the large amount of 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 135 permissions as "dangerous". The list of dangerous permissions can be used as a

guideline to help identify malicious applications. 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, 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 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 our approach dynamically determines significant permission based on actual usage by the applications instead of statically defining dangerous permission based on their intended services. This difference allows our approach to detect more malware than the approach that uses the dangerous list alone. We also compare the accuracy and running-time performance of our approach against DREBIN [2], PERMISSION-INDUCED RISK MALWARE DETECTION [14], and existing virus scanners. Again, we find that our approach can detect more malware than the other approaches with low overhead.

In summary, our paper makes the following 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 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. We also evaluate generality of SIGPID by employing 67 other machine learning algorithms. When we evaluate the malware detection effectiveness of SIGPID, we find that our approach is more effective by detecting 93.62% of malicious apps in the data set and 91.4% unknown malware.

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

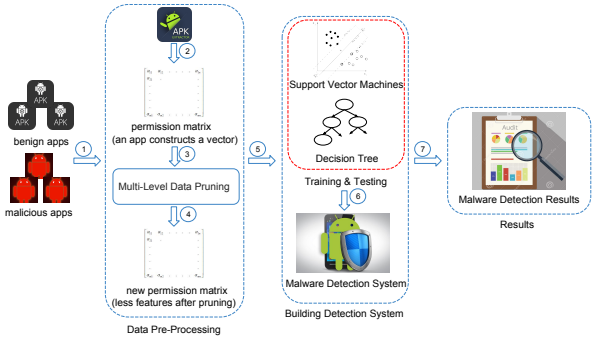



**Figure 1. Architectural Overview of SigPID**

## 2 Introducing SigPID

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 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: (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.

### 2.1 Multi-Level Data Pruning (MLDP)

A first component of SIGPID is the multi-level data pruning process to identify only significant permissions instead of considering all 135 permissions available in Android. 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.

We use 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 procedure is illustrated in Figure 2. We then describe each level in the pruning process.

#### 2.1.1 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 [2] [14], and we do 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 malicious 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. 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 [14]. 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.

Note that 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 balanced. Training over imbalanced dataset can lead to skewed models [8]. 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:

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

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:

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

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 simply take the absolute value of each number and the result of $|R(P_j)|$ ranges between [0, 1].

Next, we design a *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. We then evaluate malware detection by using the following metrics: precision, recall(true positive rate), false positive rate, accuracy, and F-measure. 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 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. After applying PRNR, we find that using 95 permissions performs nearly as well as using the whole 135 permissions. Results of our evaluation is provided in Section 3.

#### 2.1.2 Support Based Permission Ranking (SPR)

To further reduce the number of permissions, we turn our focus to the support of each permission. Typically, if

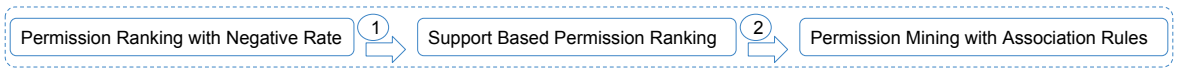



**Figure 2. Multi-Level Data Pruning**

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_SERVICE 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 310,926 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 do so, we use PIS to find the least number of permissions with high support while yielding good accuracy. After applying SPR, we are able to reduce the number of significant permissions to just 25 or 19% of the total permissions. Again, the results of our evaluation is provided in Section 3.

#### 2.1.3 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 reduce 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 both also belong to the "dangerous" permission list provided by Google. As such, 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 apply permission mining with association rules (PMAR) [1]. In all, we are able to remove three additional permissions, giving us 22 permissions that we consider as significant.

## 3 Evaluation

In this section, we evaluate the malware detection effectiveness of the SIGPID system. 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 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 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.

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). 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 contribute by each level of permission mining procedure. In addition, we also evaluate the runtime efficiency of multi-level data pruning. Mainly, 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 permission ranking methods such as Mutual Information [14] as well as signature based malware detection commonly employed by commercial virus scanners.

### 3.1 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 [2], 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. Imbalanced dataset can result in skewed models [8]. We also create another collection containing 1,661 malware samples [14] with no overlap with the initial sets. This new collection is used to evaluate the effectiveness of our system to detect newer unknown malware.

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

**Table 1. Rankings by PRNR and Mutual Information**

| MLDP | | Mutual Information | |
|---|---|---|---|
| 22 Permissions | | Top 22 Permissions | |
| ACCESS_WIFI_STATE | READ_LOGS | READ_SMS | WRITE_CALL_LOG |
| CAMERA | READ_PHONE_STATE | WRITE_SMS | VIBRATE |
| CHANGE_NETWORK_STATE | READ_SMS | SEND_SMS | CHANGE_NETWORK_STATE |
| CHANGE_WIFI_STATE | RECEIVE_BOOT_COMPLETED | WRITE_APN_SETTINGS | DEVICE_POWER |
| DISABLE_KEYGUARD | RESTART_PACKAGES | RECEIVE_SMS | WRITE_SETTINGS |
| GET_TASKS | SEND_SMS | INSTALL_PACKAGES | ADD_SYSTEM_SERVICE |
| INSTALL_PACKAGES | SET_WALLPAPER | READ_PHONE_STATE | ACCESS_NETWORK_STATE |
| READ_CALL_LOG | SYSTEM_ALERT_WINDOW | READ_EXTERNAL_STORAGE | ACCESS_LOCATION_EXTRA_COMMANDS |
| READ_CONTACTS | WRITE_APN_SETTINGS | RESTART_PACKAGES | WAKE_LOCK |
| READ_EXTERNAL_STORAGE | WRITE_CONTACTS | RECEIVE_BOOT_COMPLETED | ACCESS_COARSE_LOCATION |
| READ_HISTORY_BOOKMARKS | WRITE_SETTINGS | WRITE_CONTACTS | GET_ACCOUNTS |

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 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. We report the results in Table 2. 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 Fault Positive Rate (FPR) of over 8.5% (column 5).

**Table 2. 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 to determine the number of significant permissions after each pruning technique is applied, we present Fig. 3, which shows the key performance metrics of using PRNR with PIS as we gradually vary the number of permissions. We also present Fig. 4, which shows the standard deviation of key performance metrics of using 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 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, 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. After pruning 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.

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 [14]. It uses 40 permissions and achieves 86.4% TPR and 1.4% FPR. We list these permissions in Table 1.

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 app (higher

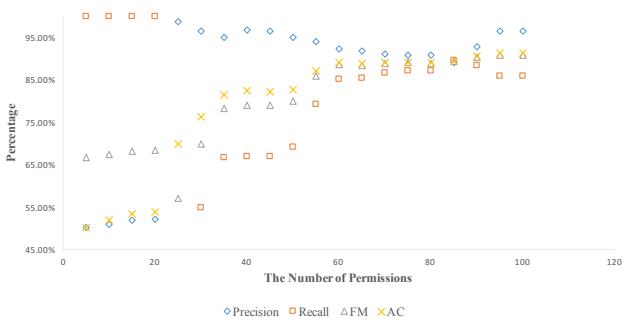



**Figure 3. Malware Classification Performance of Permission Incremental System with PRNR**

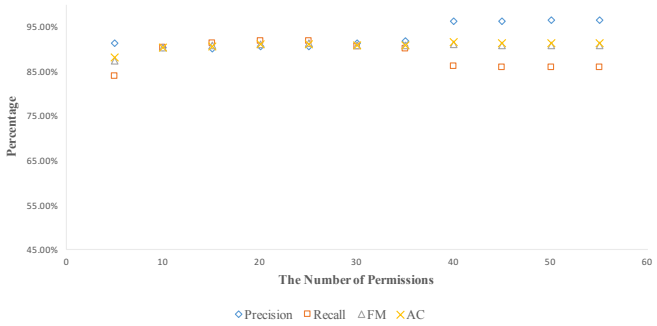



**Figure 4. Malware Classification Performance of Permission Incremental System with SPR**

recall rate), which is an important property of an effective malware detector.

### 3.3 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 [7]. 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. Table 3 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.

**Table 3. 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% |

In Table 4, we have the average processing times of the five top performing machine learning algorithms, based on recall rates of using 22, 40 [14], 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.

**Table 4. 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% |

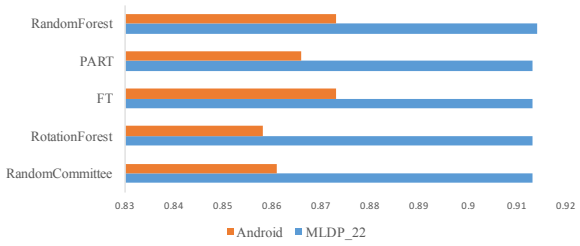



**Figure 5. Detection Performance using Unknown Real-World Malware**

We then apply our approach and malware detection based on the list of dangerous permissions to detect a different collection of apps containing 1,661 real-world malware. Again, we apply the top five machine learning algorithms for this evaluation. We report the result in Figure 5, which shows that our approach consistently detects 4% to 5.5% more malware than the approach that uses 24 dangerous permissions.

**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 incur by an approach that analyzes the complete

**Table 5. Detection Rates of SIGPID and Anti-Virus Scanners**

| SigPID w/ FT | SigPID w/ SVM | Mutual Information [14] | Drebin [2] | 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% |

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. While this experiment is still somewhat limited, it does provide some evidence that the approach can be used to effectively detect malware beyond the initial data set that we used for training and testing.

### 3.4 Comparison with Other Approaches

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

DREBIN [2] 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 [14] 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 5. 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 [2]. 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 call 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 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.

## 4 Related Work

Previous research efforts have used Android permissions to detect malware. Huang et al. [9] 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, Naive 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 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 [2] 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. [14] 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 3.4. Moreover, our work needs less permissions compared to this work, but a high detection rate can still be achieved.

## 5 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 1000 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 scanner. It can detect 93.62% of malware in the data set, and 91.4% unknown 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] R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.

[2] 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.

[3] G. DATA. G data mobile malware report. 2015, Accessed at Sep. 6, 2016.

[4] 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)*, 32(2):5, 2014.

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

[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*, pages 281–294. ACM, 2012.

[7] 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*, 11(1):10–18, 2009.

[8] H. He and E. A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, Sept 2009.

[9] 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*, pages 111–120. Springer, 2013.

[10] I. IDC Research. Smartphone os market share, 2015 q2. In *IDC Research Report*, 2015.

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

[12] 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*, page 5. ACM, 2014.

[13] Symantec. Latest intelligence for march 2016. In *Symantec Official Blog*, 2016.

[14] 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*, 9(11):1869–1882, 2014.