Significant Permission Identification for Machine-Learning-Based Android Malware Detection

Jin Li, Lichao Sun, Qiben Yan, Zhiqiang Li, Witawas Srisa-an, and Heng Ye

***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 s. 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 Significant Permission IDentification (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 three 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 a support vector machine 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 4–32 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 dataset and 91.4% unknown/new malware samples.**

***Index Terms*—Access control, computer security, machine learning algorithms, mobile applications, mobile computing.**

Manuscript received October 2, 2017; revised December 5, 2017; accepted December 12, 2017. Date of publication January 12, 2018; date of current version July 2, 2018. This work was supported in part by the National Science Foundation under Grant CNS-1566388, in part by the Defense Advanced Research Projects Agency under Grant FA8750-14-2-0053, and in part by the Maryland Procurement Office under Grant H98230-14-C-0140. This work was also supported by the Natural Science Foundation of Guangdong Province for Distinguished Young Scholars under Grant 2014A030306020, the National Natural Science Foundation of China under Grant 61472091, and the National Natural Science Foundation for Outstanding Youth Foundation under Grant 61722203. 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. J. Li and L. Sun contributed equally to this paper. Paper no. TII-17-2324. *(Corresponding author: Qiben Yan.)*

J. Li is with the School of Computer Science and Educational Software, Guangzhou University, Guangzhou 510006, China (e-mail: jinli71@gmail.com).

L. Sun is with the Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607 USA (e-mail: lsun@cse.unl.edu).

Q. Yan, Z. Li, and W. Srisa-an are with the Department of Computer Science and Engineering, University of Nebraska–Lincoln, Lincoln, NE 68588 USA (e-mail: qyan@cse.unl.edu; zli@cse.unl.edu; witty@cse.unl.edu).

H. Ye is with the Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University, Beijing 100044, China (e-mail: heng.ye@bjtu.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2017.2789219

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 s [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]–[18]. For example, RiskRanker [6] utilizes static analysis to discover malicious behaviors in Android apps. However, static analysis approaches generally assume that 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 the 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 a potential problem [19]–[21]. Petra *et al.* [22], [23] show that there is another broad range of antianalysis techniques that 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 [24]–[26]. 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 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" [27]. At the 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 Significant Permission IDentification (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 multilevel data pruning (MLDP) approach including *permission ranking with negative rate* (PRNR), *permission mining with association rules* (PMAR), and *support-based permission ranking* (SPR) 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 a 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 eight permissions jointly appear on our list and their list. This is because, as a data-driven approach, SIGPID dynamically determines significant permissions based on the 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 [28], 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. Compared 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 (5494 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 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 MLDP to reduce analysis efforts. Our system consists of three major components, designed based on real-time data analysis: PRNR, SPR, and PMAR. After pruning, SIGPID employs

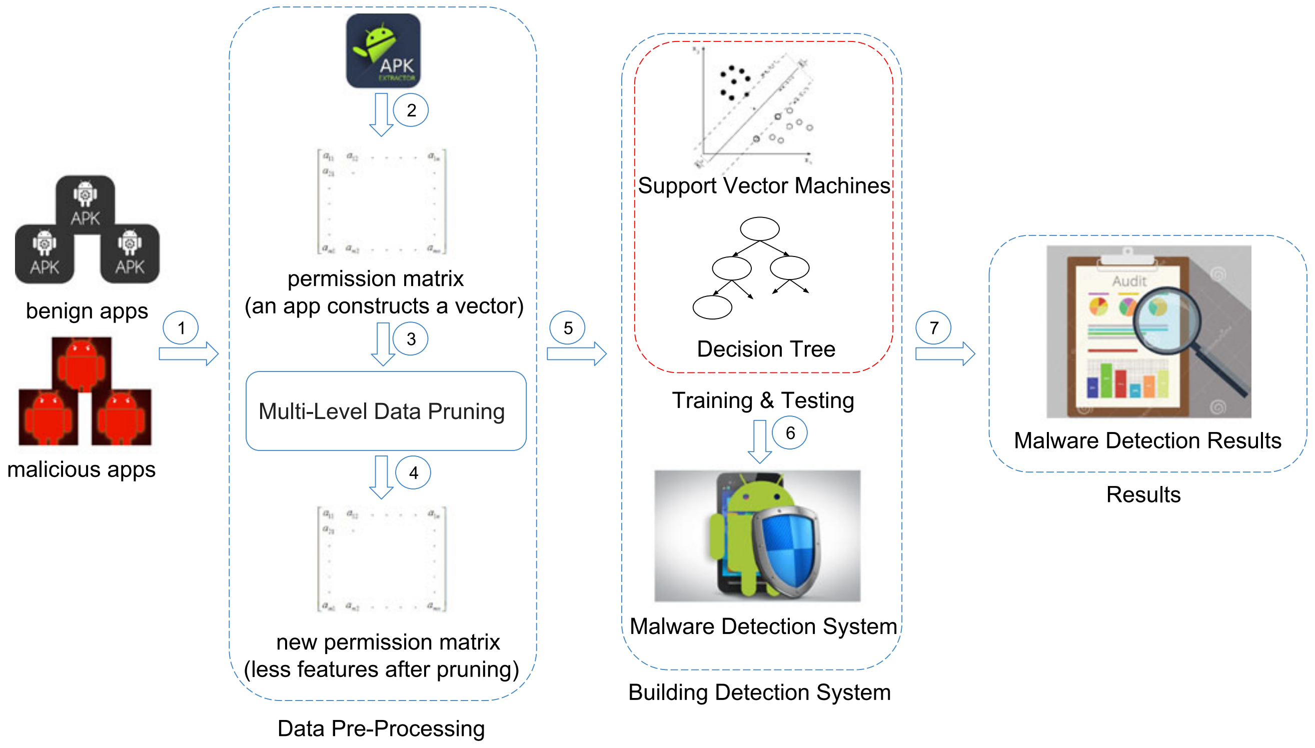

Fig. 1. Architectural overview of SIGPID.

supervised machine learning classification methods to identify potential Android malware. Finally, SIGPID reports a malware detection summary to the analysts. The complete system architecture of SIGPID is shown in Fig. 1. We then describe the key components in the remainder of this section.

A. Multilevel Data Pruning

The first component of SIGPID is the MLDP 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.

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 Fig. 2. We then describe each level in the pruning process.

1) Permission Ranking With Negative Rate: Each permission describes a particular operation that an app is allowed to perform. For instance, permission INTERNET indicates whether the app has access to the Internet. Different types of benign apps and malicious apps may request a variety of permissions corresponding to their operational needs. For malicious apps, we hypothesize that their needs may have common subsets [9], [28], 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 a generic permission ranking strategy such as mutual information to identify high-risk permissions [28]. 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 that can help detect the malware, while

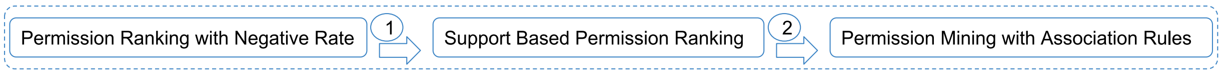
Permission Ranking with Negative Rate
1
Support Based Permission Ranking
2
Permission Mining with Association Rules

Fig. 2. Multilevel data pruning.

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 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 jth permission is requested by the ith malware sample, while "1" indicates yes and "0" indicates no. B_{ij} represents whether the jth permission is requested by the ith 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 (5494 samples in this paper); therefore, the size of B is much larger than the size of M. With our ranking scheme, we prefer the dataset on the two matrices to be balanced. Training over an imbalanced dataset can lead to skewed models [29]. To balance the two matrices, we use (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

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

where P_j denotes the jth permission and $S_B(P_j)$ represents the support of the jth permission in matrix B. PRNR can then be implemented as

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

The PRNR algorithm is used to perform ranking of our datasets. In the formula above, $R(P_j)$ represents the rate of the jth 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 the malicious dataset, which is a high-risk permission. If $R(P_j) = -1$, this means that permission P_j is only used in the benign dataset, which is a low-risk permission. If $R(P_j) = 0$, this means that P_j has a 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 ($\frac{2P*R}{P+R}$). Then, we choose the top three permissions in both lists to build the 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 dataset. Within our evaluation dataset with 5494 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 are provided in Section III.

2) Support-Based Permission Ranking: 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 the 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 permission 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 the 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: After pruning 110 of 135 permissions by using PRNR and SPR with the PIS, we want to further explore approaches that can remove noninfluential 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 PMAR mechanism using the association rule mining algorithm [30].

Association rule mining has been used for discovering meaningful relations between variables in large databases. For instance, if event A and B always cooccur, 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 [30] 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 three 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 the SVM and a small dataset to test our proposed MLDP model. The 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 the 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 the 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 relies on a training dataset to build a classifier in deciding which class the testing data is in. Note that building the classifier requires a significant preprocessing effort, especially when the training dataset contains many attributes. Therefore, the pruning of attributes in the decision tree model is imperative, which can be supported by the proposed MLDP.

Consequently, it is more advantageous to use MLDP to perform malware detection as it can be effective while notably conserving time and memory. Since the time and memory are limited in common computers, we can outsource the task to cloud in a suitable way to boost efficiency. Because the permissions can be recorded perfectly by using a binary vector, we could remove the mapping table and do 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 the 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: PRNR, SPR, and PMAR. We evaluate the malware detection performance by enabling these multiple levels sequentially to verify the performance improvement contributed by each level of the permission mining procedure. In addition, we also evaluate the runtime efficiency of MLDP.

TABLE I
Detection Results Using SVM With MLDP 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

In most of the experiments, the 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 [28] as well as signature-based malware detection commonly employed by commercial virus scanners.

A. Collected Dataset

We generate ten datasets by randomly choosing 5494 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 the balance during training, as the imbalanced dataset can result in skewed models [29]. We also create another newer collection of apps in June 2015 containing 2650 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 downloaded from both Google Play and Anzhi Store from January 2012 to December 2014 [28]. We compared our experiments using three different datasets, as will be shown in Section III-D. 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

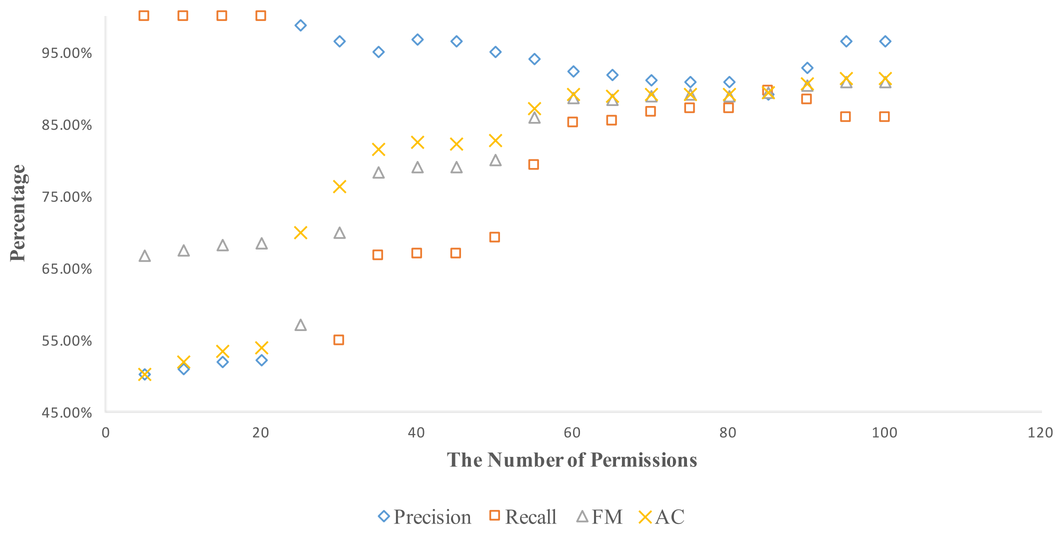

Fig. 3. Malware classification performance of the PIS with PRNR.

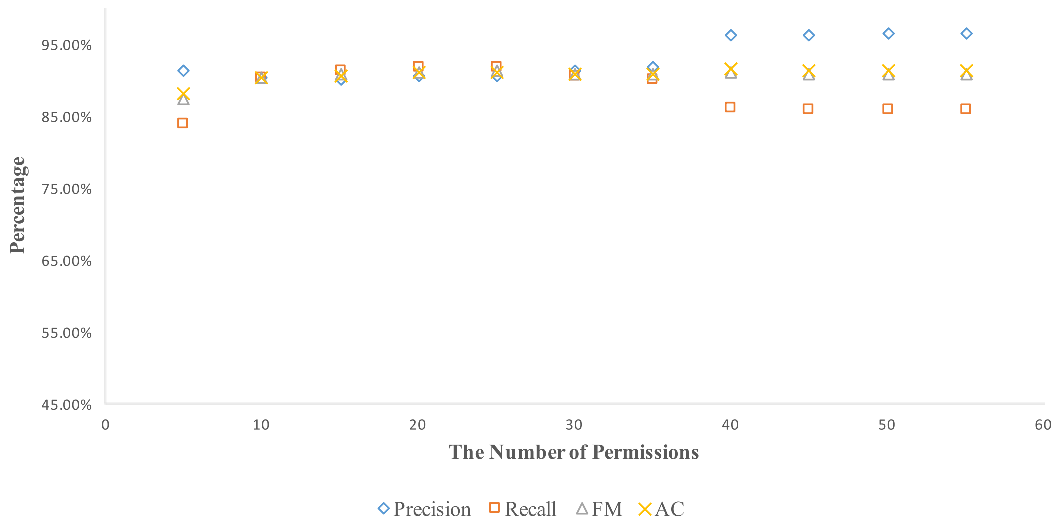

Fig. 4. Malware classification performance of the PIS with SPR.

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 MLDP 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 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 FPR of over 8.5% (column 5).

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 Figs. 3 and 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 PMAR to perform the last layer of permission mining. Here, we find three rules satisfying our

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

associative requirements when we set our confidence level to 96.5%. The association rule 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 coocccurred/associated. After pruning these three 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.

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 [28]. 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 eight 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 [31]. 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 five 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 reports 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 FPR is also 1.09% higher, which is reasonable considering the improvement in other evaluation metrics.

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 [28], 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 s on average, compared to 24.55 s 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 2650 real-world malware. Again, we apply the top five machine learning algorithms for this evaluation. We report the result in Fig. 6, which shows that our approach consistently detects 4–5.5% 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 four times (when rotation forest is used) to 32 times (when FT 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 the tree struc-

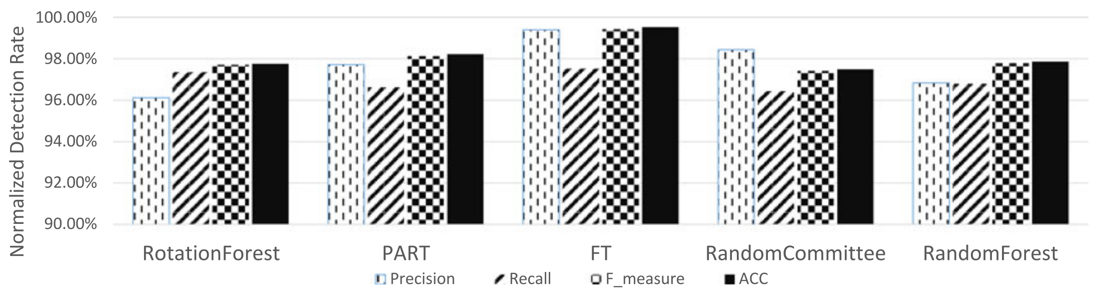

Fig. 5. Results with top five machine learning algorithms.

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%

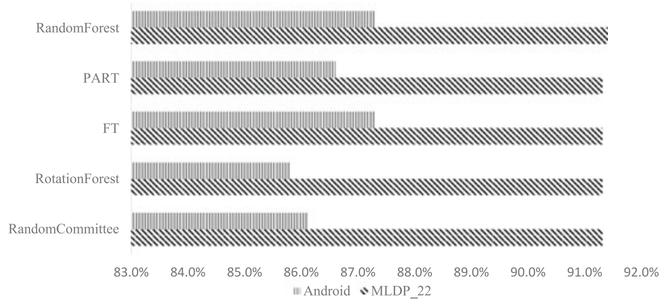

Fig. 6. Detection performance using unknown real-world malware.

ture can produce better results. Among the top ten efficient algorithms, five 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 dataset that we used for training and testing, which shows the potential applicability of SIGPID in practice.

TABLE V
TESTING RESULTS USING DIFFERENT DATASETS

Num_of_Malapp	2650	5494	54694
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%

D. Applicability of SigPID

As described before, we have collected three different real-world datasets, which contain 2650, 5494 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 that the size of the dataset is huge.

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 a dataset based on permissions and other features from apps. It then utilizes the SVM algorithm to classify the 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 [28] 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 dataset, and the ratio of their malicious and benign apps in their dataset is dominated by benign apps. As such, their reported results, especially FPR, are significantly different than the results achieved using our dataset.

The comparison results are shown in Table VI. DREBIN uses more features than our SIGPID, including application programming interface calls and network addresses. As a result, DREBIN outperforms PERMISSIONCLASSIFIER in detection

TABLE VI
DETECTION RATES OF SIGPID AND ANTIVIRUS SCANNERS

SigPID w/FT	SigPID w/SVM	Mutual information [28]	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%

accuracy. We also compare the results against ten existing antivirus 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 nonrisky permissions (e.g., READ_CALL_LOG), which are only used in benign apps and have high support values. We deem the risky and nonrisky 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 the existing malware scanners 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.

IV. Related Work

Previous research efforts have used Android permissions to detect malware. Huang *et al.* [32] 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 [33], 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.* [28] 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* and *principal component analysis*. 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% FPR. Their low FPR 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 three-level pruning approach. Based on our dataset, which includes over 2000 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 dataset and 91.4% unknown/new malware.

Acknowledgment

The authors would like to thank the anonymous reviewers for their insightful feedback on this work.

References

[1] IDC, "Smartphone OS market share, 2017 q1," 2017. [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," May 2016. [Online]. Available: https://www.statista.com/statistics/281106/number-of-android-app-downlo ads-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." 2017. [Online]. Available: https://www.gdatasoftware.com/blog/2017/04/29712-8-400-new-android-malw are-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 *Proc. 10th Int. Conf. Mobile Syst., Appl., Services*, 2012, pp. 281–294.

[7] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proc. 18th ACM Conf. Comput. Commun. Security*, 2011, pp. 627–638.

[8] W. Enck *et al.*, "TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Trans. Comput. Syst.*, vol. 32, no. 2, 2014, Art. no. 5.

[9] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, K. Rieck, and C. Siemens, "DREBIN: Effective and explainable detection of android malware in your pocket," presented at Annu. Symp. Netw. Distrib. Syst. Security, 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 *Proc. Eur. Symp. Res. Comput. Security*, 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 *Proc. 3rd Int. Workshop Building Anal. Datasets Gathering Exp. Returns Security*, 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 *Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng.*, 2015, vol. 1, 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," in *Proc. IEEE Conf. Comput. Commun. Workshops*, 2017, pp. 18–23.

[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 *Proc. Int. Conf. Security Privacy Commun. Syst.*, 2016, pp. 597–616.

[15] Z. Chen *et al.*, "Machine learning based mobile malware detection using highly imbalanced network traffic," *Inf. Sci.*, Apr. 2017, doi: https://doi.org/10.1016/j.ins.2017.04.044.

[16] S. Wang, Q. Yan, Z. Chen, B. Yang, C. Zhao, and M. Conti, "Detecting android malware leveraging text semantics of network flows," *IEEE Trans. Inf. Forensics Security*, vol. PP, no. 99, pp. 1–1, 2017.

[17] J. Z. L. H. P. S. Y. Lichao Sun, X. Wei, and W. Srisa-an, "Contaminant removal for android malware detection systems," presented at IEEE Int. Conf. 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 *Proc. Joint Eur.Conf. Mach. Learn. Knowl. Discovery Databases*, 2017, pp. 228–240.

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

[20] 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 Gener. Comput. Syst.*, vol. 74, pp. 76–85, 2017.

[21] P. Li, J. Li, Z. Huang, C. Gao, W. Chen, and K. Chen, "Privacy-preserving outsourced classification in cloud computing," *Cluster Comput.*, pp. 1–10, 2017, doi: 10.1007/s10586-017-0849-9.

[22] T. Petsas, G. Voyatzis, E. Athanasopoulos, M. Polychronakis, and S. Ioannidis, "Rage against the virtual machine: Hindering dynamic analysis of android malware," in *Proc. 7th Eur. Workshop Syst. Security*, 2014, Art. no. 5.

[23] 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 Trans. Ind. Informat.*, vol. 13, no. 1, pp. 186–197, Feb. 2017.

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

[25] K. Bu, M. Xu, X. Liu, J. Luo, S. Zhang, and M. Weng, "Deterministic detection of cloning attacks for anonymous RFID systems," *IEEE Trans. Ind. Informat.*, vol. 11, no. 6, pp. 1255–1266, Dec. 2015.

[26] T. Cruz *et al.*, "A cybersecurity detection framework for supervisory control and data acquisition systems," *IEEE Trans. Ind. Informat.*, vol. 12, no. 6, pp. 2236–2246, Dec. 2016.

[27] G. Android, "Requesting permissions." 2017. [Online]. Available: https://developer.android.com/guide/topics/permissions/requesting.html

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

[29] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, pp. 1263–1284, Sep. 2009.

[30] R. Agrawal *et al.*, "Fast algorithms for mining association rules," in *Proc. 20th Int. Conf. Very Large Data Bases*, 1994, vol. 1215, pp. 487–499.

[31] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The Weka data mining software: An update," *ACM SIGKDD Explorations Newslett.*, vol. 11, no. 1, pp. 10–18, 2009.

[32] 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*, vol. 2. New York, NY, USA: Springer, 2013, pp. 111–120.

[33] 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 Trans. Ind. Informat.*, vol. 12, no. 3, pp. 1005–1016, Jun. 2016.

Authors' photographs and biographies not available at the time of publication.