Android Malware Detection via (Somewhat) Robust Irreversible Feature Transformations

Qian Han, V. S. Subrahmanian, and Yanhai Xiong

***Abstract*— As the most widely used OS on earth, Android is heavily targeted by malicious hackers. Though much work has been done on detecting Android malware, hackers are becoming increasingly adept at evading ML classifiers. We develop FARM, a Feature transformation based AndRoid Malware detector. FARM takes well-known features for Android malware detection and introduces three new types of feature transformations that transform these features irreversibly into a new feature domain. We first test FARM on 6 Android classification problems separating goodware and "other malware" from 3 classes of malware: rooting malware, spyware, and banking trojans. We show that FARM beats standard baselines when no attacks occur. Though we cannot guess all possible attacks that an adversary might use, we propose three realistic attacks on FARM and show that FARM is very robust to these attacks in all classification problems. Additionally, FARM has automatically identified two malware samples which were not previously classified as rooting malware by any of the 61 anti-viruses on VirusTotal. These samples were reported to Google's Android Security Team who subsequently confirmed our findings.**

***Index Terms*— Android, machine learning, feature transformation, malware detection, spyware, Banking Trojans, rooting malware.**

I. Introduction

THE Android platform is the most widely used operating system in the world today [24].[1] Because it is both very popular and open source, it is subject to a wide variety of attacks [1]–[3] involving theft of credentials, bank fraud, click fraud, ransomware, adware, SMS fraud, and more.

While there has been extensive work on Android malware analysis and detection, [7], [12], [13], [15], [21], [38], [40], we know that when a malware is detected by anti-virus vendors or white hats, the malicious hackers involved try to evade the signature. Though no one can predict all the ingenious evasion methods that malicious hackers might come up with in the future, it is important that malware detectors try to be robust to evasion methods.

We propose FARM (short for Feature transformation based AndRoid Malware detector), a framework for detecting Android Rooting Malware which is robust to certain types of attacks that we might expect malicious hackers to try out. Given a set of standard features used in the literature for Android malware prediction in general, our first major contribution is a set of three *new* and *broad* classes of feature transformations that irreversibly map the original feature space into a new feature space.

- *Landmark based transformations.* Our first class of feature transformations is based on the idea of landmarks. The basic idea is similar to triangulation: every point in the original feature space can be characterized by its distance from a given set of landmarks. Because the selection of distance functions and landmark points is flexible, this is not one transformation, but an entire class of transformations. The adversary will have difficulty figuring out what a specific implementation of FARM is doing, even if they know all the original features.
- *Feature clustering based transformations.* Our second innovation is a class of transforms that clusters sets of similar features together by inspecting the original features. If two features have similar values in the original training data, then those two features should end up in the same cluster. Each cluster of features generates a new feature (one per cluster) having a value computed from the feature values in that cluster. This too is not one transformation, but a class of transformations because FARM can use one of any number of clustering algorithms and pick whatever reasonable hyper-parameters it feels are appropriate for that clustering algorithm. This is also hard for an adversary to guess, even if they know the entire training set.
- *Correlation graph based transformations.* This is an idea similar to that above. The original features in the training data end up as nodes in an undirected graph. An edge linking two features is weighted by a selected correlation coefficient (e.g. Pearson's correlation coefficient). The features are then clustered together and each feature-cluster corresponds to a new feature in a manner similar to that in the preceding transformation.

Our second major contribution is an extensive set of experiments that combine the transformed features with standard classifiers and a late fusion step. Specifically, we test the FARM approach on 6 Android malware classification problems: (i) rooting apps vs. goodware, (ii) rooting apps vs. other malware, (iii) spyware vs. goodware, (iv) spyware vs. other malware, (v) banking trojans vs. goodware, and (vi) banking trojans vs. other malware. Each of the 6 Android malware classification problems is tested on two types of data: one

Manuscript received July 5, 2019; revised November 29, 2019 and January 27, 2020; accepted February 6, 2020. Date of publication February 26, 2020; date of current version June 26, 2020. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Wei Yu. *(Corresponding author: V. S. Subrahmanian.)*

The authors are with the Department of Computer Science, Institute for Security Technology and Society, Dartmouth College, Hanover, NH 03755 USA (e-mail: vs@dartmouth.edu).

Digital Object Identifier 10.1109/TIFS.2020.2975932

[1] https://www.c-sharpcorner.com/article/what-is-the-most-popular-operating-system/

TABLE I

PERCENTAGE IMPROVEMENT OF FARM OVER THE BEST BASELINE ON F1-SCORE

Classification Problem	Improvement
Goodware vs. Rooting (No-Isomorphic)	6.02%
Goodware vs. Rooting (Isomorphic)	4.38%
Goodware vs. Banking Trojan (No-Isomorphic)	3.92%
Goodware vs. Banking Trojan (Isomorphic)	1.60%
Goodware vs. Spyware (No-Isomorphic)	2.64%
Goodware vs. Spyware (Isomorphic)	1.46%
Other-malware vs. Rooting (No-Isomorphic)	2.80%
Other-malware vs. Rooting (Isomorphic)	2.27%
Other-malware vs. Banking Trojan (No-Isomorphic)	7.21%
Other-malware vs. Banking Trojan (Isomorphic)	6.43%
Other-malware vs. Spyware (No-Isomorphic)	2.12%
Other-malware vs. Spyware (Isomorphic)	1.05%

where no samples in the data are isomorphic, and another where only one copy of isomorphic samples is retained in the data.[2] Thus, in total, there are 12 sets of experiments that we conduct to assess the performance of FARM.

When no attacks are present, our 10-fold cross-validation experiments show that FARM achieves an improvement in the F1-score of 1.05-6.43% over the baseline classifiers. The percentage improvement[3] of FARM vs. the best baseline is summarized in Table I. This suggests that in the absence of attacks, FARM outperforms strong baselines – though not by a huge margin.

However, when certain types of attacks on classifiers occur, FARM strongly outperforms the baselines, yielding our third major contribution. Malicious hackers continuously look for methods to evade FARM. We propose three potential attacks and evaluate the robustness of FARM w.r.t. these attacks. Our 10-fold cross-validation experiments show that FARM is more robust in the face of these attacks than past work.[4] An impact score less than 1 says that FARM is more robust against the attack than the baselines. The smaller the impact score, the better. Moreover, for example, an impact score of 0.2 under a given attack says that the adverse impact on FARM is 20% of the adverse impact on the baseline, i.e. FARM is 5 times more resilient than the baseline. We see in Tables VI and VII that the impact score of FARM under the first two attacks is consistently lower than 1. This is also the case with the third kind of attack (Table VIII). Here, it is worth noting that the impact score of FARM actually goes into negative territory, suggesting that FARM actually performs better under the third type of attack than when there is no attack. This is counter-intuitive and we will discuss why later in Section VI. Tables VI, VII and VIII below respectively show the impact of the three types of attacks on all 12 problems in the case of the baseline vs. FARM. We see that FARM is often many times

[2]Two Android samples are considered isomorphic if they have the same feature vector.

[3]Percentage improvement of FARM over the best baseline for classification problem $\wp$ is given by the formula $\frac{F1(\text{FARM})}{F1(Baseline)} - 1$.

[4]The *impact score* of attack a on FARM is given by $\frac{RedF1(\text{FARM},a)}{RedF1(baseline,a)}$ where $RedF1(Alg, a)$ is the reduction in F1 score of algorithm Alg when the attack a happens, i.e. $RedF1(Alg, a) = F1Score(Alg, \text{no attack}) - F1Score(Alg, \text{attack a})$

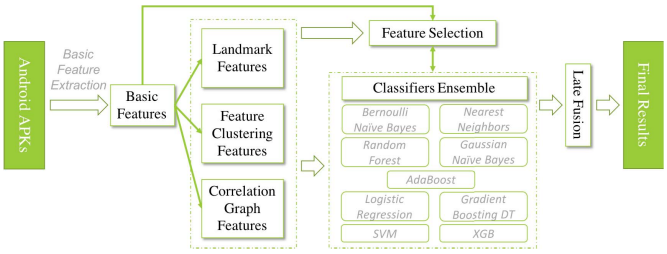

Fig. 1. The FARM Framework.

better than the baseline in all of these cases. On the first type of attack, we see that impact score of FARM is between 10.47% to 72.12%, i.e. FARM varies may be up to almost 10 times more resilient to this attack than the baseline. On the second type of attack, FARM's impact score varies from 10.25% to 74.42%, showing almost a similar range of greater robustness than the baseline. On the third type of attack, FARM's impact score ranges from -3.9 to +6.14, suggesting the FARM, while often much more robust, is not always more robust. Overall, of the 36 situations tested (12 times 3 tables), FARM's superior robustness was established in 35 of 36 cases. *That said, we do not claim that FARM is robust against all kinds of attacks - just the three types of attacks proposed in this paper.*

Finally, we note that FARM has found 2 malware samples that were previously not known to be rooting malware to any of the 61 anti-virus engines on VirusTotal (as well as to Google). We reported these two samples to Google's Android Security team who confirmed the findings.

Figure 1 shows the architecture of the FARM framework — this paper is organized in a manner that is consistent with this architecture. Section II discusses related work on Android malware detection. Our FARM dataset consisting of Android goodware, rooting malware, spyware, banking trojans, and other malware is discussed in Section III. As we build on top of basic features used in other work, we only provide a brief description of those in Section IV. Section V contains a comprehensive description of the three feature transformation techniques (landmark features, feature clustering features, correlation graph features) used in this paper. Experimental results are presented and discussed in Section VI.

II. RELATED WORK

We discuss 2 types of related work: work on Android malware detection in general, and work on feature transformation techniques for machine learning.

A. Literature on General Android malware Detection Techniques

We discuss two categories of related work here: (i) static analysis based detection of known malicious patterns in source code and other relevant metadata, and (ii) dynamic analysis which tests an Android APK by executing it in real-time and monitoring the results. Previous examples of malware detection solutions include MaMaDroid [26], SigPID [21], MADAM [31], the deep android malware detection system [27], Hindroid [17], DREBIN [7], DroidAPIMiner [4], CrowDroid [11], DroidScope [39] and MARVIN [23].

1) Static Analysis: DREBIN [7] and DroidAPIMiner [4] use lightweight features based on static analysis to distinguish Android malware. DREBIN detects 94% of the malware with a low false positive rate while DroidAPIMiner achieves 99% accuracy and 2.2% false positive rate. MaMaDroid [26] builds a classifier to detect malware based on features extracted from a behavioral model in the form of a Markov chain capturing the sequence of API calls performed by an APK. [35] provides a systematic study of permission-induced risk in APKs. Their results show that with the top 40 risky permissions, the detection rate with Random Forest reaches 0.9462 with a false positive rate of 0.006. SigPID [21] develops methods to identify significant permissions related to Android malware prediction. They show that only 22 out of 135 permissions are needed to achieve over 90% detection accuracy. Reference [27] uses static analysis and convolutional neural networks to separate Android malware from goodware. Hindroid [17] develops a heterogeneous information network and then uses meta-path analysis to predict if a sample is benign or malicious. [12] develops a regression model based on decompiled code analysis to distinguish malware and goodware. Reference [42] proposes an end-to-end method for automatic feature engineering by mining documents written in natural language. The results achieve a 92.5% true positive rate with only 1% false positive rate, which is comparable to the detectors based on manually engineered features. Reference [33] detects repackaged Android malware via code heterogeneity analysis. They partition the code into multiple dependence-based regions and each region is classified independently based on behavioral features. Reference [22] performs Android malware family clustering efficiently with novel malicious payload mining techniques.

2) Dynamic Analysis: CrowDroid [11] detects malicious applications that have benign names and versions. It also collects and compares execution traces from users using a crowdsourcing approach. DroidScope [39] analyses APKs that collect native, Dalvik instruction traces and profiles API-level activity. It also tracks information leakage through taint analysis. MARVIN [23] uses machine learning to distinguish between goodware and malware with static and dynamic analysis, achieving a detection accuracy of 98.24%. MADAM [31] develops methods using features at four levels: kernel, application, user and package levels to detect malicious behaviors, achieving over 96% accuracy and a low false positive rate. DroidTrace [41] implements a dynamic analysis system by using ptrace to monitor the selected system calls of the target process, then executing classification according to their sequence. Reference [28] develops AuntieDroid based on MaMaDroid and Chimp [5] (a crowdsourced method) to detect malware with both static and dynamic analysis, yielding F-measure of 0.92. XManDroid [10] examines the use of transitive permissions in Android's inter-process communication protocol in order to detect privilege escalation attacks. Reference [38] describes the Pileup vulnerability using which malware can declare a set of privileges and attributes in an older version of the operating system until the system is updated to a newer version. They show that attackers can attack thousands of devices from different manufacturers, carriers, and countries. They develop a detector that scans devices to capture exploits based on the Pileup vulnerability. RootExplorer [15] studies the challenging problem of finding root exploits in a different way. They consider the fact that there are now commercial grade "root providers" [40] including major corporations such as Tencent, Baidu, and Qihoo who provide this as a service so naive users can take certain actions that require root privileges such as removing bloatware. Well-known Android malware guru Romain Unuchek points out that "Users rooting their own devices offer quite a gift to malware developers" [34] as malicious hackers can use the code provided by root providers to launch attacks. RootExplorer looks at root exploits provided by such "root providers" and tries to detect them effectively. Although not aimed for malware detection, we note that CopperDroid [32] designs an automatic virtual machine introspection based dynamic analysis system to reconstruct the OS and Android-specific behaviors of Android malware which could be leveraged for malware detection.

FARM differs from these prior malware detection efforts in two broad respects: first, FARM focuses on developing a method for malware detection that is robust in the presence of various types of attacks while the above efforts do not (with the exception of [8]. Additionally, FARM looks at 12 classification problems in all spanning 3 different types of Android malware (rooting malware, spyware, and banking trojans). FARM develops 3 novel feature transformations for these purposes and shows that these feature transformations lead to greater robustness under certain types of adversarial attacks.

To evaluate the robustness of the proposed feature transformation techniques, we also investigated related literature on attacks on defensive methods. Reference [8] studies behaviors of defenses relying on obfuscated gradients and how they can be circumvented. The partition method is used by [25] to replace a piece of malware with a number of "shallow processes" to evade detection by system-call behavior based detectors. Reference [30] use common obfuscation methods to generate attack malware samples. [18]'s MalGAN system generates malware samples from a single feature vector. In particular, they add irrelevant features to avoid detection of the original malware. Reference [16] selects features via optimization and adds them to malware samples as a kind of attack. FARM use three attack models to modify malware samples.

B. Literature on Feature Transformation Techniques for Machine Learning

Most work on feature transformation applies feature transformation techniques to reduce computational complexity or improve object recognition accuracy. At the very outset, we note that we are not aware of any efforts to use feature transformation for Android malware prediction, nor are we aware of any feature transformation efforts directed at evading malware variants.

Reference [29] evaluates the use of PCA-based feature transformation and shows that in some cases, it can yield better

TABLE II
DATASET DESCRIPTION

Number of APKs	Isomorphic	No-Isomorphic
Goodware	3535	2999
Rooting Malware	1829	444
Banking Trojans	7107	1061
Spyware	3247	841
Other-Malware (Not Rooting)	4596	2081
Other-Malware (Not Banking Trojans)	3973	1806
Other-Malware (Not Spyware)	4382	1922

performance than feature selection. Reference [36] proposes an adaptive conformal transformation (ACT) algorithm in order to achieve better classification results when training data is imbalanced. Cognito [20] proposes the use of transformation trees for improved feature engineering. Reference [6] provides evidence to show that feature transformation may lead to improved predictive accuracy. Reference [19] designs an incremental matrix factorization framework using a linear feature transformation of user and item latent vectors, showing a relatively high accuracy and space-efficient training process in an online scenario. Reference [37] proposes a convex radius-margin-based SVM model for joint learning of feature transformation and an SVM classifier, and shows that it outperforms both classical SVM and some advanced SVM-based methods. [14] presents a novel semi-supervised learning framework to improve visual classification performance using a sequence of feature transformations.

In contrast to past work on feature transformations, FARM proposes three entirely new classes of feature transformation techniques that can be widely adapted for different machine learning models. In addition, FARM introduces very realistic potential attacks by adversaries and shows that it is robust to these attacks — past work on feature transformations do not present any insights on adversarial evasion.

III. THE FARM DATASET

In this section, we briefly introduce the FARM dataset which consists of a mix of Android goodware, rooting malware, spyware, banking trojans, and other malware. For a sample to be tagged in one of these malware categories, we required that there be at least 2 reports on Koodous[5] confirming this status. Table II summarizes the statistics of the FARM dataset.

We say that two APKs are isomorphic if they have the same set of API features. While the description of the "standard" features used by us appears in the next section, we note that using isomorphic samples for 10-fold cross-validation is fundamentally wrong because two malware samples (with different hashes) may end up with the same feature vectors. If one of these is in the training set and one in the validation set, then the testing protocol is severely compromised and will inflate the predictive accuracy results. We therefore have two versions of our dataset: the *No-Isomorphic* version has no isomorphic samples in it, while the *With-Isomorphic* dataset allows the isomorphic samples to persist. We report our main experimental results on both versions of the FARM dataset for the sake of completeness and clarity. We asked a few cybersecurity experts why the number of malware samples that are isomorphic is so large. The reason seems to be that malware developers build a piece of malware and deploy it - but at some point in time, cybersecurity firms develop signatures to detect it. At this point, the malware developer usually tweaks his malware slightly to evade the signature, and then the cybersecurity firm tweaks its signature. This process keeps iterating many times – at least for profitable malware samples – thus leading to many malware samples with identical feature vectors.

[5]https://koodous.com/

IV. BASIC FEATURES

As the "basic features" associated with Android APKs are not a contribution of this paper and are derived from past work [9], [13], we describe them very briefly here.[6] The basic features fall into 3 categories.

A. Static Features

We use Androguard to extract 120 static features such as APK size, developer information, statistics (on the number of activities, message receivers, providers and functionalities,) and one-hot encoded permission features.

B. API Package Call Features

We also consider a recent class of lightweight API-based package-level static features introduced in 2019 [9]. In the Android system, API packages contain one or more API classes. API packages may interact with the operating system and provide basic communication services, e.g., *android.os* provides basic operating system services, message passing and internal communication between processes on the device. Figure 2 illustrates the relationship between API packages, API classes, and API methods. For example, API package *android.os* contains API class *android.os.Debug*, *android.os.Message* and *android.os.UserManager*, etc. Similarly, each API class contains one or more API methods. As in the case of [9], FARM has features that capture the frequency with which an API class in a given API package was called. *We do not consider API method call frequencies because it these frequencies are very expensive to compute in terms of time at the method level - in contrast, the package level computations are relatively fast.*

C. Dynamic Features

We use Koodous' Cuckoo and Droidbox based analysis to extract 767 dynamic features including: files written, files read, DNS connected, crypto usage, SMS activities, phonecall activities, library activities, dex calls, etc. We extract the statistics of these features (e.g., the total number of times crypto operators are used) and also construct a one-hot coding for categorical features.

[6]A complete description of the basic features used can be viewed at https://docs.google.com/spreadsheets/d/1StlowS2Zm25MtLsx_xIvfdqZiyYrb2_f1TVVKVkrV1I/edit?usp=sharing

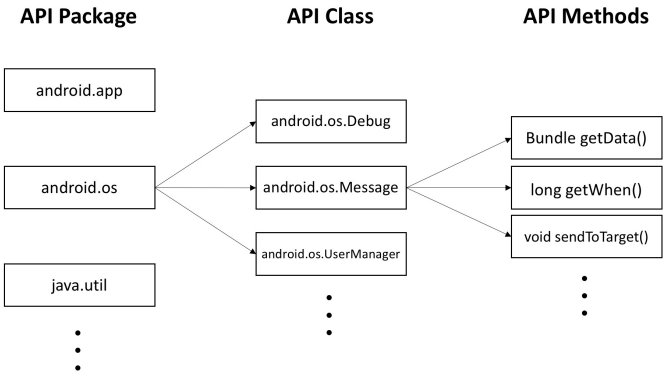

Fig. 2. Android API Package call features illustration.

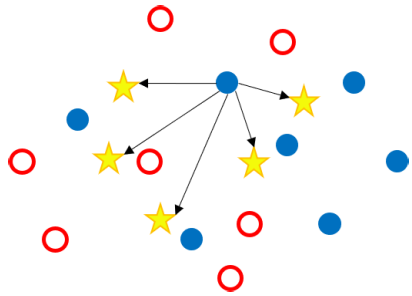

Fig. 3. Illustration of landmark based feature transformation. Red and blue dots represent a type of malware (e.g. spyware) and goodware in the feature space respectively. Yellow stars stand for selected landmarks.

FARM contains 171 API package call features — and in all, FARM associates a 1058-dimensional feature vector with each APK. We call this the "basic feature vector" of an APK.

Training Set. Throughout this paper, we assume a training set consisting of m APKs with feature vectors $f_1, \ldots, f_m$ respectively. We assume that there are n basic features in all and hence we can represent the feature vectors as a feature table $F = \{f_{ij}\}_{1 \le i \le m, 1 \le j \le n}$ where the rows correspond to feature vectors of APKs and each column corresponds to a feature. Of course, for each of the 12 classification problems we study in this paper, each APK i has an associated class $y_i \in \{0, 1\}$. So when we look at the problem of separating rooting malware from goodware, we set $y_i = 0$ to mean that sample i is goodware, while $y_i = 1$ means it is a rooting app. When we look at the problem of separating spyware from goodware, we set $y_i = 0$ to mean that sample i is goodware, while $y_i = 1$ means it is spyware.

V. Feature Transformation Techniques

FARM does not directly use the basic features defined in the previous section for prediction. It starts by first selecting a subset $F \subseteq F_{base}$ of the basic features. The feature vectors w.r.t. F_{base} are then restricted to F.[7]

FARM transforms the restricted feature vectors associated with each APK to a new feature space using a set of 3 irreversible transformations designed to: (i) keep accuracy of prediction comparable to the case when the basic features are used, and (ii) be robust against adversarial attempts to evade the classifier. The latter property makes it more difficult for attackers to adjust the attributes of their malware to evade detection.

A. Landmark Based Feature Transformation

Consider the US map. Every point on the US map is characterized by a vector consisting of an apartment number (possibly nil), street number, street name, town, state, zip code, and possibly even more features. However, we might choose a set of say 5 landmark points in the US. We can then map each original feature vector to a new vector of length 5 where each entry in the new vector is the distance from that entry to one of the landmark points. This 5-dimensional vector is likely to represent the original point on the map. Thus, the street address "236 Riverside Drive Apt 5A, Manhattan, NY 10025" which precisely identifies an apartment may now be represented by the 5-d feature vector (20, 11, 216, 492, 117). In this example, the distance from the above Manhattan address to the first landmark point is 20, the distance to the second landmark point is 11, and so forth. An adversary who merely sees this feature vector cannot reconstruct the original address unless he knows: (i) the 5 landmark points used *and* (ii) the distance function used. We build upon this simple intuition to map the basic feature vectors via landmark based transformations.

Figure 3 is a simple illustration of landmark based feature transformation. Assume there is a set of rooting malware (red circles) and goodware (blue dots) represented in the basic feature space (for the ease of visualization, we present them in a 2-dimension space instead of an n-dimension space where n is the number of basic features). With a set of landmarks (yellow stars) selected, for each APK (red circle or blue dot), we can compute a new feature vector for it according to its distance to each landmark. Formally, suppose there are LM landmarks selected from m samples in the dataset. Then for each sample i in the dataset we get a new feature vector $f_{LM} = (d_{i,1}, \cdots, d_{i,LM})$, where $d_{i,j}$ is the distance from sample i to landmark j. More details about generating LM features are presented in Algorithm 1.

Algorithm 1: Generating LM Features

Input: $F' = \{f_{ij}\}_{1 \le i \le m, 1 \le j \le n}$ (n-dimensional basic feature vectors for m sample APKs), LM, landmark selection method λ and distance measure $dist$;
Select landmarks $\ell_1, \ldots, \ell_{LM}$ from m samples by applying landmark selection method $\lambda(F)$ to F;
for *each sample APK* i **do**
  for *each landmark* ℓ_h **do**
    $d_{i,\ell_h} = dist(i, \ell_h)$;
  $\mathbf{f}_i^{LM} = (d_{i,\ell_1}, \cdots, d_{i,\ell_{LM}})$ % landmark feature vector
return $F_{LM} = \{\mathbf{f}_i^{LM} \mid 1 \le i \le m\}$ (landmark based LM-dimensional feature vectors for m sample APKs);

[7]Because we do not want the adversary to easily guess what features we are using, F is best selected in a random manner so the adversary has difficulty in guessing what was chosen. Alternatively, it could also be selected by using the N features that generate the best predictive results. These will also be hard for the adversary to guess as he will need to have the same training set in order to make a guess.

In addition to the set F of basic feature vectors, Algorithm 1 needs three more parameters to generate LM features. They are discussed in detail below.

1) The number of landmarks LM. While LM can be any integer from 1 to m, in our experiments, we vary LM over the set $\{3, 6, 9, \ldots, 27, 30\}$.
2) A distance measure $dist(i, \ell_h)$ between sample i's basic feature vector and landmark ℓ_h's basic feature vector can be one of many standard measures. We use Euclidean distance, Manhattan distance, Cosine distance, and Hamming distance in our experiments though of course other distances metrics may be used as well.
3) A method to select landmarks. We examine three ways to do this.
   - Random selection. LM landmarks are randomly selected from the training data.
   - $k-$means clustering based selection. The training samples are first clustered using $k-$means clustering with $k = LM$. One sample is then randomly selected from each cluster as the landmark.
   - Max-distance heuristic selection. Algorithm 2 shows an algorithm for selecting landmarks that are scattered across the basic feature space. This algorithm starts by randomly choosing a training point as a landmark and then iteratively adding training points. In each iteration, a random sample of training points is drawn and the point that is "furthest away" (in terms of the sum of its distance) from the previously selected landmarks is the next choice. The process ends when LM landmarks have been picked.

Algorithm 2: Max-Distance Landmark Selection

Input: Set M with m samples, $F = \{f_{ij}\}_{1 \le i \le m, 1 \le j \le n}$ (n-dimensional basic feature vectors for m training APKs), LM, and distance measure $dist$;
$\ell_0 =$ Randomly select one training sample from M
$SOL = \{\ell_0\}$
while $|SOL| < LM$ **do**
  $R =$ random samples drawn from $M - SOL$
  $Best = \arg\max_{r \in R} \Sigma_{\ell \in SOL} dist(l, r)$
  $SOL = SOL \cup \{Best\}$
return SOL

B. Feature Value Clustering Based Feature Transformation

We now move on to our second feature transformation which is based on the intuition that similar features may be combined together to make a smaller but perhaps more representative set of features. Algorithm 3 presents the method we use to do this.

Our FC-feature generation algorithm is also pictorially depicted in Figure 4 works as follows. It considers each column in the feature table (corresponding to a basic feature) to be a column vector. We see the different feature columns shown on the left of Figure 4. It then uses a clustering algorithm to cluster the column vectors into G groups. Thus,

Algorithm 3: Generating FC Features

Input: $F = \{f_{ij}\}_{1 \le i \le m, 1 \le j \le n}$, G: number of clusters desired, Clu a clustering algorithm, $\oplus$ associative and commutative feature combination algorithm;
Cluster the n basic features into G groups accordingly by considering each feature to be a column vector in F;
for *each sample APK* i **do**
  for *each feature group* g **do**
    $f_{ig}^{FC} = \oplus\{i.f \mid f \in g\}$ % combine values of APK i's value of feature f for each f in feature group g;
  $\mathbf{f}_i^{FC} = (f_{i1}^{FC}, \cdots, f_{iG}^{FC})$ % FC feature vector for sample i;
return $F_{FC} = \{\mathbf{f}_i^{FC} \mid 1 \le i \le m\}$ (feature value clustering based G-dimensional feature vectors for m sample APKs);

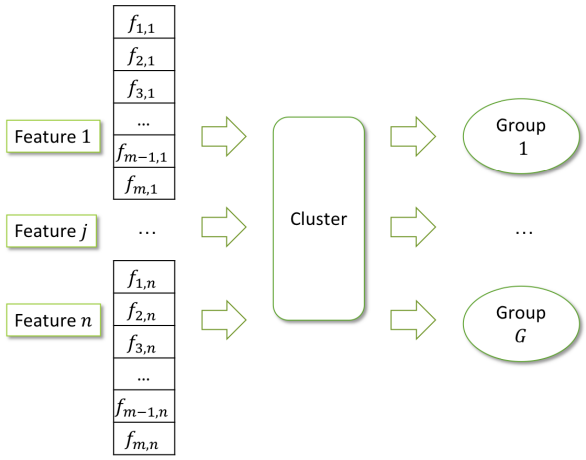

Fig. 4. Feature Value Clustering (FC) of features into G groups.

each group consists of a set of features which are similar enough to be clustered together by the clustering algorithm. The FC-feature transformation associates just one "merged" feature for each group. For any given APK sample i, the value of the feature associated with a specific group g ($1 \le g \le G$) of features is obtained by first computing the set $\{i.f \mid f \in g\}$ and then combining all the feature values in this set into one by using an associative and commutative combination operator $\oplus$ which is part of the input to the algorithm. As $\oplus$ is associative and commutative, the order in which it combines the members of the set $\{i.f \mid f \in g\}$ does not matter. The result is a new G-dimensional feature vector for each APK sample i.

We consider three possible definitions of $\oplus$ in our experiments though our algorithm works with any possible associative and commutative operator. Specifically, we consider:

1) Product of the group of features as a new feature: $\oplus(X) = \prod_{x \in X} x$.
2) Average of the group of features as a new feature: $\oplus(X) = (\sum_{x \in X} x)/|X|$.
3) Distance-inverse weighted sum of the group of features as a new feature: $\oplus(X) = \alpha \sum_{x \in X} x \times e^{-d(x,\mu)}$ where μ is the centroid of X and $dist$ is a distance measure.

Note that we try all distance measures stated for landmark based feature transformation.

It is important to note that the FC-transformation above maps each APKs n-dimensional basic feature vector to a G-dimensional space which would usually be much smaller. The FC-transformation makes several choices. One is the choice of the clustering algorithm to use — but in addition, a "hidden" choice is the choice of parameters to use in the clustering algorithm. Another example is the choice of number of clusters. A third is the choice of $\oplus$. And even within $\oplus$, a fourth choice is the choice of parameters within $\oplus$. As an example, suppose we had 100 features to start with and the FC-transformation used $G = 3$. In this case, every APK has a new FC-feature vector with just 3 values (x_1, x_2, x_3). An adversary looking at this would have difficulty even knowing which of the original features went into generating x_1, which went into generating x_2 and which went into generating x_3, let alone know the answers to the additional choices mentioned above. *An adversary who reads this paper would still have considerable difficulty in determining how all of these choices were made in a real-world implementation of* FARM.

C. Correlation Graph Based Feature Transformation

We now come to our third feature transformation which also tries to divide the n basic features into a number of groups using the novel concept of a correlation graph (CG). We first construct a symmetrical matrix with n rows and n columns, where c_{j_1,j_2}, the j_2th element in the j_1th row, represents the correlation of feature j_1 and feature j_2. We can also think of this as a graph whose nodes are features and where an edge (undirected) linking features j_1, j_2 is weighted by the Pearson Correlation Coefficient between the two features. We then cluster these n features would be clustered into G groups, and we generate a new feature vector $\mathbf{f}_{CG}$ for each sample APK in the dataset.

Algorithm 4: Generating CG Features

Input: $F = \{f_{ij}\}_{1 \le i \le m, 1 \le j \le n}$, Clu a clustering algorithm, G desired number of groups, $\oplus$ an associative and commutative operator;
Compute $n \times n$ correlation matrix according to column vectors of F;
Cluster the n basic features into G groups according to rows of correlation matrix;
for *each sample APK* i **do**
  for *each feature group* g **do**
    $f_{ig}^{CG} = \oplus\{i.f \mid f \in g\}$;
  $\mathbf{f}_i^{CG} = (f_{i1}^{CG}, \cdots, f_{iG}^{CG})$ CG feature vector for sample i;
return $F_{CG} = \{\mathbf{f}_i^{CG} \mid 1 \le i \le m\}$ *(correlation graph based G-dimensional feature vectors for m sample APKs)*;

As in the case of the FC-transformations, the correlation graph transformation looks at rows of the correlation matrix and clusters features together based on the correlations that exist. The remainder of the algorithm generates a smaller set of features as was the case with FC-transformations.

Important Note. The three feature transformations described here may apply to a wide variety of classification problems. In this paper, we show that they lead to no compromise (in fact an improvement) over baselines for 12 Android malware classification problems, and we additionally show that they are resilient to certain types of adversarial attacks. We believe that these results are likely to hold for many other domains as well, but we do not assert this as a claim of the paper. We further do not claim robustness against all types of adversarial attacks, just the three mentioned in this paper.

VI. Experimental Evaluation

In this section, we describe the results of the experiments that we have designed to evaluate the performance of FARM with different feature combinations. Our experimental evaluation includes four parts: (1) Distinguishing each of the 3 types Android malware (banking trojans, rooting malware, spyware) from goodware on both the Isomorphic and No-Isomorphic datasets; (2) Distinguishing each of the 3 types Android malware (banking trojans, rooting malware, spyware) from *other types* of malware on both the Isomorphic and No-Isomorphic datasets; (3) Evaluating robustness of FARM against 3 attacks. In addition, FARM discovered two new rooting malware samples - a fact that was not previously known to any of the 61 anti-virus engines on VirusTotal. *As (1) and (2) involve 12 experiments in all, we present a sample in the main body of the paper. Readers may find more details at*[8] *part of the paper.* We used 10−fold cross-validation and 8 classifiers.[9]

Late Fusion. The predicted probabilities p_i of the $M = 8$ classifiers $C_i, i = 1, \ldots, 8$ are linearly combined by FARM as $p = \sum_{i=1}^{M} \gamma_i p_i$, where $\sum \gamma_i = 1$. We find the best value of the γ_is by doing a grid search and optimizing performance on the training set.

A. No Adversarial Attack Case

Tables III and IV summarize the results of experiments on 5 settings (described below) in the *no attack* case. Each of the 5 settings is defined below.[10] LM for LM, FC, CG for FC and CG. When more than one of LM, FC and CG are used at the same time, their Ns are set to the same value. "Distance" column is used to state the best distance measure

[8] https://drive.google.com/open?id=14ZQyFtsu6exZhoav4z-1aDZXPWrnMNjv

[9] Classifiers used: (1) Bernoulli Naive Bayes, (2) Random Forest, (3) Nearest Neighbors, (4) Logistic Regression, (5) Gaussian Naive Bayes, (6) AdaBoost Classifier, (7) Gradient Boosting Decision Tree, (8) XGB Classifier and (9) SVM.

[10] We use "SD" to refer to static and dynamic features, "API" to refer to API package call features, "LM" to refer to the landmark based features (furthermore, we use "-Rand", "-Cluster" and "-Max-dis" to represent the three types of landmark selection methods), "FC" corresponds to feature value clustering based features, and "CG" is for correlation graph based features. We use "LF$(\cdot, \ldots, \cdot)$" denotes the late fusion classifier with appropriate feature inputs. Of the various metrics reported, the most important one is the "F1-score", which reflects a balance of precision and recall. The column N stands for the number of landmarks or clusters used by LM, FC and CG.

TABLE III

MULTIPLE METRICS (AUC AND F1 ETC.) ON ANDROID MALWARE DETECTION VS. GOODWARE / OTHER-MALWARE NO-ISOMORPHIC DATASET

	Datasets	Best Settings	N	Distance	Classifier	AUC	Precision	Recall	F1	FPR	FNR
SET 1 Baseline	Goodware vs. Rooting	SD + API	-	-	RF	0.9943	0.9932	0.8776	0.9312	0.0067	0.0231
	Goodware vs. Banking Trojan	SD + API	-	-	RF	0.9974	0.9869	0.9306	0.9577	0.0131	0.0241
	Goodware vs. Spyware	SD + API	-	-	RF	0.9949	0.9697	0.9697	0.9697	0.0303	0.0033
	Other-malware vs. Rooting	SD + API	-	-	RF	0.9937	0.9817	0.8636	0.9181	0.0182	0.0261
	Other-malware vs. Banking Trojan	SD + API	-	-	RF	0.9498	0.8835	0.8949	0.8887	0.1165	0.1070
	Other-malware vs. Spyware	SD + API	-	-	RF	0.9922	0.9582	0.9733	0.9657	0.0418	0.0293
SET 2 FARM w/ LM	Goodware vs. Rooting	LF(LM-Cluster, SD, API)	12	Hamming	RF	0.9983	1.0000	0.9534	0.9761	0.0000	0.0060
	Goodware vs. Banking Trojan	LF(LM-Cluster, SD, API)	30	Hamming	GBDT	0.9952	1.0000	0.9565	0.9778	0.0000	0.0068
	Goodware vs. Spyware	LF(LM-Cluster, SD, API)	24	Hamming	RF	0.9972	0.9903	0.9697	0.9798	0.0097	0.0320
	Other-malware vs. Rooting	LF(LM-Cluster, SD, API)	30	Hamming	RF	0.9826	0.9241	0.9521	0.9378	0.0759	0.0534
	Other-malware vs. Banking Trojan	LF(LM-Cluster, SD, API)	18	Hamming	GBDT	0.9614	0.9352	0.9619	0.9484	0.0648	0.0388
	Other-malware vs. Spyware	LF(LM-Cluster, SD, API)	21	Hamming	RF	0.9974	0.9915	0.9738	0.9826	0.0085	0.0277
SET 3 FARM w/ FC	Goodware vs. Rooting	LF(FC, SD, API)	27	-	RF	0.9990	1.0000	0.9705	0.9850	0.0000	0.0030
	Goodware vs. Banking Trojan	LF(FC, SD, API)	30	-	XGB	0.9949	1.0000	0.9444	0.9714	0.0000	0.0066
	Goodware vs. Spyware	LF(FC, SD, API)	21	-	RF	0.9969	0.9910	0.9709	0.9808	0.0090	0.0307
	Other-malware vs. Rooting	LF(FC, SD, API)	21	-	RF	0.9696	0.9090	0.9331	0.9208	0.0910	0.0743
	Other-malware vs. Banking Trojan	LF(FC, SD, API)	30	-	XGB	0.9669	0.8991	0.9515	0.9245	0.1009	0.0490
	Other-malware vs. Spyware	LF(FC, SD, API)	24	-	RF	0.9978	0.9856	0.9865	0.9860	0.0144	0.0147
SET 4 FARM w/ CG	Goodware vs. Rooting	LF(CG, SD, API)	21	-	RF	0.9978	1.0000	0.9512	0.9750	0.0000	0.0060
	Goodware vs. Banking Trojan	LF(CG, SD, API)	30	-	RF	0.9987	0.9904	0.9810	0.9856	0.0096	0.0066
	Goodware vs. Spyware	LF(CG, SD, API)	30	-	RF	0.9982	0.9910	0.9831	0.9870	0.0090	0.0181
	Other-malware vs. Rooting	LF(CG, SD, API)	27	-	RF	0.9849	0.9221	0.9595	0.9403	0.0779	0.0457
	Other-malware vs. Banking Trojan	LF(CG, SD, API)	21	-	RF	0.9687	0.9115	0.9537	0.9321	0.0885	0.0510
	Other-malware vs. Spyware	LF(CG, SD, API)	30	-	RF	0.9978	0.9825	0.9867	0.9846	0.0175	0.0145
SET 5 FARM w/ all	Goodware vs. Rooting	LF(LM, FC, CG, SD, API)	30	Hamming	RF	0.9950	1.0000	0.9750	0.9873	0.0000	0.0033
	Goodware vs. Banking Trojan	LF(LM, FC, CG, SD, API)	27	Euclidean	RF	0.9992	1.0000	0.9905	0.9952	0.0000	0.0033
	Goodware vs. Spyware	LF(LM, FC, CG, SD, API)	21	Hamming	RF	0.9991	1.0000	0.9906	0.9953	0.0000	0.0098
	Other-malware vs. Rooting	LF(LM, FC, CG, SD, API)	21	Hamming	RF	0.9847	0.9545	0.9333	0.9438	0.0454	0.0147
	Other-malware vs. Banking Trojan	LF(LM, FC, CG, SD, API)	24	Euclidean	RF	0.9689	0.9487	0.9569	0.9528	0.0513	0.0532
	Other-malware vs. Spyware	LF(LM, FC, CG, SD, API)	27	Hamming	RF	0.9981	0.9840	0.9885	0.9862	0.0160	0.0125

for classifiers with LM features, and "Classifier" stands for the classifier selected with the best performance.

*1) **SET 1** Baseline: Basic Features Only:* Due to the large number of SD features, we first compared the performance of classifiers with all or part of SD features using feature selection methods. We found that a certain number of selected features yielded the best F1 score. This is done via a standard ablation test. In ablation testing, we first compute the performance (F1-score) with all features; we then drop 1 feature and see which feature leads to the biggest drop in performance — this feature, f_1 is the most important. We then repeat this process to find the second most important feature f_2 (which is the feature that leads to the biggest drop in performance, assuming f_1 is already dropped), the third most important feature f_3, and so forth. For each f_j, we compute the performance of the classifiers using the features in $F_j = F - \{f_1, \ldots, f_j\}$. For each F_j, we compute the performance of our classifiers using just the features in $F - F_j$, and choose the j that leads to the highest performance. When distinguishing between rooting malware and goodware, we found that $j = 50$ selected features lead to the best F1 score 0.9195. We then trained classifiers with API features only, as well as the combination of SD and API features (row "SD + API") respectively. The results of combining SD and API features (better than using SD or API features alone) in Tables III and IV (SET1) show that the baselines achieve F1-scores of 88.87-96.97% and 91.96-98.26% on the No-Isomorphic and Isomorphic datasets respectively. These are the numbers that FARM has to beat.

*2) **SET 2** FARM w/ LM: FARM With Landmark Based Features:* Our SET 2 experiments first compared FARM with LM-features alone while changing the landmark selection method and varying the number of landmarks LM. Of the three landmark selection methods, we found that the max-distance heuristic selection (LM-Max_dis) is both not competitive and far more time-consuming. We therefore abandoned this method in the following experiments. Next, we compared the remaining two landmark selection methods by combining them with SD, API and SD + API features respectively. The results show that FARM with landmark features alone beats the baseline *in all 12 cases* with F1-Scores of 93.78-98.26% and 94.06-99.08% for the No-Isomorphic and Isomorphic datasets respectively.

*3) **SET 3** FARM w/ FC: FARM With Feature Value Clustering Based Features:* The SET 3 experiments used FARM with classifiers trained on data generated using the feature clustering based features (w.r.t. different number of clusters G and the one with best performance is presented in N column) and combine them with SD, API and SD + API features respectively. Our SET 3 results show that FARM obtains F1-scores of 92.08-98.6% and 94.22-99.06% respectively. Here again, FARM beats the baseline in all 12 experiments and returns results comparable to those generated by LM-features.

*4) **SET 4** FARM w/ CG: FARM with Correlation Graph based Features:* In SET 4 experiments, we trained our classifiers with the correlation graph based features (w.r.t. different number of clusters G) and combined them with SD, API, and SD + API features respectively. Our results show that FARM with CG features beats the baselines on all 12 cases and achieves F1-scores of 94.03-98.56% and 92.96-99.55% on the No-Isomorphic and Isomorphic datasets respectively.

*5) **SET 5** FARM w/ all: FARM Approach with All Transformed Features:* In SET 5 experiments, we used features from all the proposed feature transformation methods and combined them with SD, API and SD + API features respectively.

TABLE IV

MULTIPLE METRICS (AUC AND F1 ETC.) ON ANDROID MALWARE DETECTION VS. GOODWARE / OTHER-MALWARE ISOMORPHIC DATASET

	Datasets	Best Settings	N	Distance	Classifier	AUC	Precision	Recall	F1	FPR	FNR
SET 1 Baseline	Goodware vs. Rooting	SD + API	-	-	RF	0.9845	0.9908	0.9192	0.9535	0.0092	0.0278
	Goodware vs. Banking Trojan	SD + API	-	-	RF	0.9967	0.9903	0.9696	0.9798	0.0097	0.0321
	Goodware vs. Spyware	SD + API	-	-	RF	0.9976	0.9920	0.9734	0.9826	0.0080	0.0282
	Other-malware vs. Rooting	SD + API	-	-	RF	0.9740	0.9181	0.9525	0.9349	0.0819	0.0537
	Other-malware vs. Banking Trojan	SD + API	-	-	RF	0.9671	0.9279	0.9115	0.9196	0.0721	0.1000
	Other-malware vs. Spyware	SD + API	-	-	RF	0.9876	0.9759	0.9819	0.9788	0.0241	0.0561
SET 2 FARM w/ LM	Goodware vs. Rooting	LF(LM-Cluster, SD, API)	21	Hamming	RF	0.9985	1.0000	0.9579	0.9785	0.0000	0.0127
	Goodware vs. Banking Trojan	LF(LM-Cluster, SD, API)	27	Hamming	RF	0.9970	0.9926	0.9713	0.9818	0.0074	0.0302
	Goodware vs. Spyware	LF(LM-Cluster, SD, API)	30	Hamming	RF	0.9985	1.0000	0.9818	0.9908	0.0000	0.0067
	Other-malware vs. Rooting	LF(LM-Cluster, SD, API)	24	Euclidean	RF	0.9859	0.9284	0.9597	0.9437	0.0716	0.0453
	Other-malware vs. Banking Trojan	LF(LM-Cluster, SD, API)	27	Hamming	GBDT	0.9700	0.9500	0.9314	0.9406	0.0500	0.0631
	Other-malware vs. Spyware	LF(LM-Cluster, SD, API)	30	Hamming	RF	0.9980	0.9828	0.9881	0.9854	0.0172	0.0130
SET 3 FARM w/ FC	Goodware vs. Rooting	LF(FC, SD, API)	30	-	RF	0.9978	0.9847	0.9875	0.9861	0.0153	0.0135
	Goodware vs. Banking Trojan	LF(FC, SD, API)	24	-	RF	0.9988	1.0000	0.9813	0.9906	0.0000	0.0067
	Goodware vs. Spyware	LF(FC, SD, API)	21	-	RF	0.9987	0.9903	0.9903	0.9903	0.0097	0.0033
	Other-malware vs. Rooting	LF(FC, SD, API)	30	-	RF	0.9863	0.9242	0.9611	0.9422	0.0758	0.0438
	Other-malware vs. Banking Trojan	LF(FC, SD, API)	30	-	RF	0.9673	0.9691	0.9307	0.9495	0.0309	0.0614
	Other-malware vs. Spyware	LF(FC, SD, API)	21	-	RF	0.9981	0.9867	0.9855	0.9861	0.0133	0.0156
SET 4 FARM w/ CG	Goodware vs. Rooting	LF(CG, SD, API)	27	-	RF	0.9967	0.9903	0.9696	0.9798	0.0097	0.0321
	Goodware vs. Banking Trojan	LF(CG, SD, API)	24	-	RF	0.9987	0.9903	0.9903	0.9903	0.0097	0.0033
	Goodware vs. Spyware	LF(CG, SD, API)	30	-	GBDT	0.9991	0.9970	0.9940	0.9955	0.0030	0.0069
	Other-malware vs. Rooting	LF(CG, SD, API)	30	-	RF	0.9864	0.9255	0.9600	0.9424	0.0745	0.0449
	Other-malware vs. Banking Trojan	LF(CG, SD, API)	27	-	RF	0.9704	0.9167	0.9429	0.9296	0.0833	0.0583
	Other-malware vs. Spyware	LF(CG, SD, API)	21	-	RF	0.9979	0.9833	0.9862	0.9848	0.0167	0.0149
SET 5 FARM w/ all	Goodware vs. Rooting	LF(LM, FC, CG, SD, API)	24	Euclidean	RF	0.9991	1.0000	0.9906	0.9953	0.0000	0.0098
	Goodware vs. Banking Trojan	LF(LM, FC, CG, SD, API)	30	Hamming	RF	0.9992	0.9970	0.9940	0.9955	0.0030	0.0068
	Goodware vs. Spyware	LF(LM, FC, CG, SD, API)	24	Hamming	RF	0.9989	0.9969	0.9969	0.9969	0.0031	0.0034
	Other-malware vs. Rooting	LF(LM, FC, CG, SD, API)	21	Euclidean	GBDT	0.9884	0.9457	0.9669	0.9561	0.0543	0.0365
	Other-malware vs. Banking Trojan	LF(LM, FC, CG, SD, API)	27	Hamming	RF	0.9970	0.9903	0.9675	0.9787	0.0097	0.0343
	Other-malware vs. Spyware	LF(LM, FC, CG, SD, API)	30	Hamming	RF	0.9985	0.9815	0.9969	0.9891	0.0185	0.0095

TABLE V

STATISTICAL RESULTS P-VALUE OF BEST SETTINGS OF FARM OVER THE BEST BASELINE

Classification Problem	$p-value$
Goodware vs. Rooting (No-Isomorphic)	$1.0959e{-10}$
Goodware vs. Rooting (Isomorphic)	$1.0539e{-6}$
Goodware vs. Banking Trojan (No-Isomorphic)	$2.5891e{-7}$
Goodware vs. Banking Trojan (Isomorphic)	$1.6523e{-4}$
Goodware vs. Spyware (No-Isomorphic)	$8.1829e{-6}$
Goodware vs. Spyware (Isomorphic)	$4.4940e{-4}$
Other-malware vs. Rooting (No-Isomorphic)	$3.5337e{-3}$
Other-malware vs. Rooting (Isomorphic)	$3.3029e{-7}$
Other-malware vs. Banking Trojan (No-Isomorphic)	$7.6698e{-5}$
Other-malware vs. Banking Trojan (Isomorphic)	$1.4564e{-6}$
Other-malware vs. Spyware (No-Isomorphic)	$2.7297e{-4}$
Other-malware vs. Spyware (Isomorphic)	$3.6931e{-4}$

The experimental results show that FARM achieves F1-scores of 94.38-99.53% and 95.61-99.69%, again beating out the baselines on all 12 problems. Moreover, the combination of all three feature transformations generated the best results in all.

Statistical Significance. We tested the null hypothesis that the best baseline for each of the 12 problems considered was generated by the same underlying process as the best setting of FARM (i.e. with $LF(LM, FC, CG, SD, API)$.) The null hypothesis was rejected in all 12 cases with $p \leq$3.5337e-3 in all cases, i.e. the probability that the same underlying process generated both the best baseline results and the best FARM results is so low that it is almost zero. Thus, the claim that FARM is better than the best baseline in distinguishing across the 12 problems considered is statistically valid.

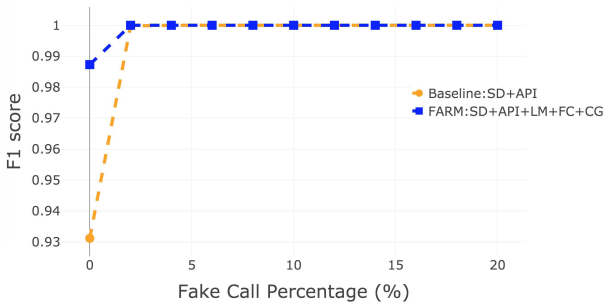

Fig. 5. Impact of fake API package call attack on Android rooting malware detection: Goodware vs. Rooting Malware (No-Isomorphic).

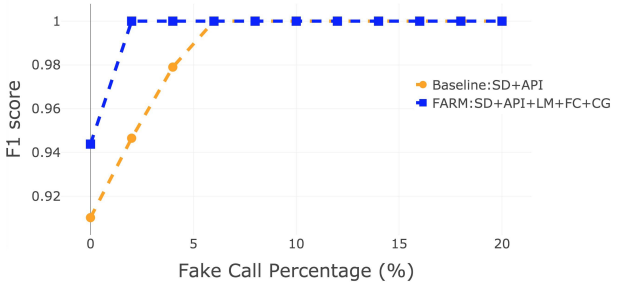

Fig. 6. Impact of fake API package call attack on Android rooting malware detection: Other Malware vs. Rooting Malware (No-Isomorphic).

B. Robustness Evaluation

The goal of the three feature transformations introduced in this paper is to make FARM more robust in the presence of adversarial attacks. We can be sure that malicious hackers will adapt their malware once they realize that it has been detected and that anti-virus engines have developed signatures to protect Android devices from the threat. Though it is impossible

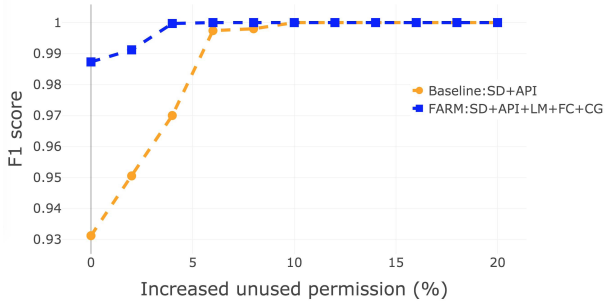

Fig. 7. Impact of increased percentage of permissions attack on Android rooting malware detection: Goodware vs. Rooting Malware (No-Isomorphic).

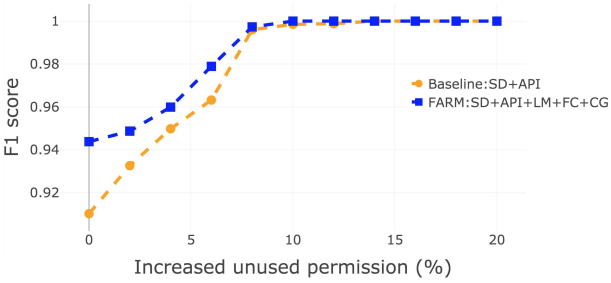

Fig. 8. Impact of increased percentage of permissions attack on Android rooting malware detection: Other Malware vs. Rooting Malware (No-Isomorphic).

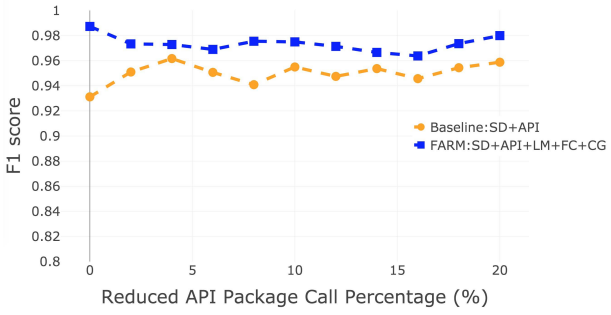

Fig. 9. Impact of reduced percentage of API package call attack on Android rooting malware detection: Goodware vs. Rooting Malware (No-Isomorphic).

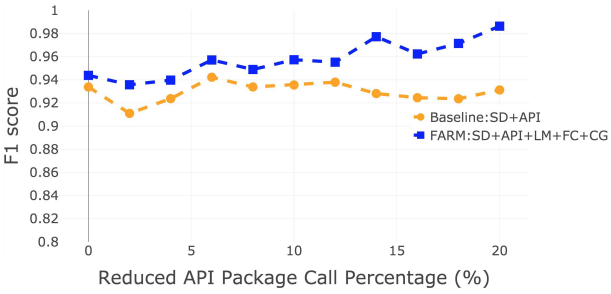

Fig. 10. Impact of Reduced Percentage of API Package Call Attack on Android Rooting malware Detection: Other Malware vs. Rooting Malware (No-Isomorphic).

to imagine all the types of evasion methods that malicious hackers might come up with, we tested the robustness of FARM against three kinds of attacks.

Threat Model. We assume that the adversary: (i) knows all the 1058 basic features used by FARM, and (ii) that the adversary is also familiar with the suite of 8 classifiers used in the paper (Bernoulli and Gaussian Naive Bayes, Random Forest, k-Nearest Neighbor, Logistic Regression, Adaboost,

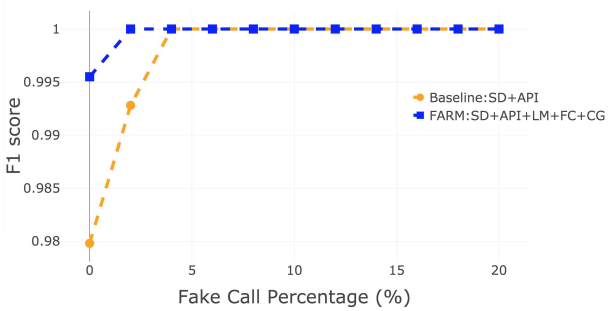

Fig. 11. Impact of fake API package call attack on Android Banking Trojans detection: Goodware vs. Banking Trojans (Isomorphic).

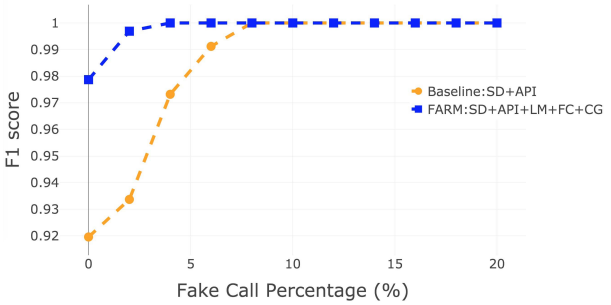

Fig. 12. Impact of fake API package call attack on Android Banking Trojans detection: Other Malware vs. Banking Trojans (Isomorphic).

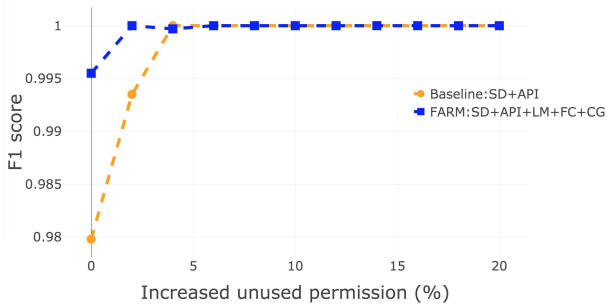

Fig. 13. Impact of increased percentage of permissions attack on Android Banking Trojans detection: Goodware vs. Banking Trojans (Isomorphic).

Gradient Boosted Decision Tree, XGB, and SVM). We further assume that the attacker has read this paper and hence knows about the three types of feature transformation used. But we do not assume the attacker knows any of the following: (i) the specific landmarks used, the landmark selection strategy used and/or distance function used by the defender in the Landmark-based Feature transformation, (ii) the number of clusters and the $\oplus$ feature combination algorithm used in the Feature-Value based Clustering Transformation, and (iii) the number of groups and the specific $\oplus$ operator used by the defender in the Correlation-Graph based feature transformation. We further assume that the attacker carries out the three kinds of attacks described below.[11] We assume the attacker tries three kinds of attacks:

1) Fake API Package calls in which the adversary injects irrelevant API package calls into his malware.

[11] We do not claim that FARM is robust against all kinds of adversarial attacks (e.g. obfuscated gradient attacks [8]). Indeed, such a claim would be very hard to justify for almost any paper without making some unrealistic assumptions.

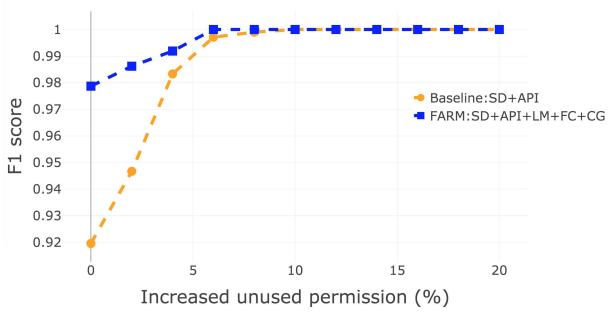

Fig. 14. Impact of increased percentage of permissions attack on Android Banking Trojans detection: Other Malware vs. Banking Trojans (Isomorphic).

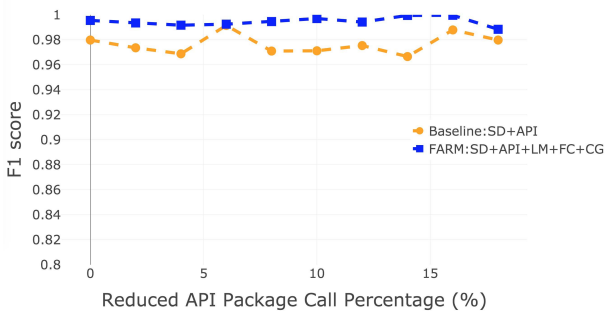

Fig. 15. Impact of reduced percentage of API package call attack on Android Banking Trojans detection: Goodware vs. Banking Trojans (Isomorphic).

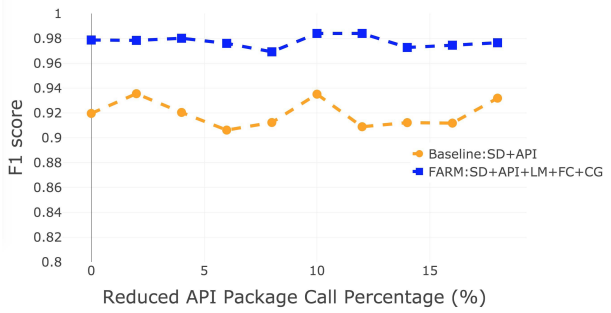

Fig. 16. Impact of reduced percentage of API package call attack on Android Banking Trojans detection: Other Malware vs. Banking Trojans (Isomorphic).

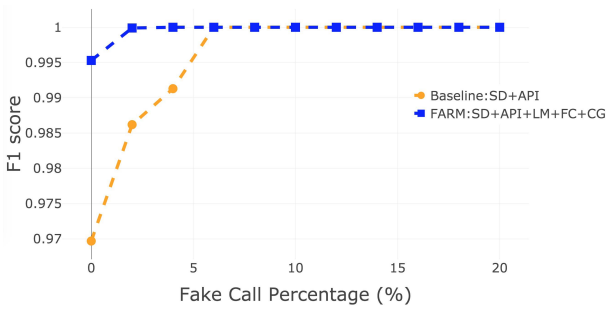

Fig. 17. Impact of fake API package call attack on Android Spyware detection: Goodware vs. Spyware (No-Isomorphic).

2) Fake permission requests in which the adversary requests permissions that are irrelevant for his malware.
3) Reduced API Package calls in which the adversary tries to artificially reduce the number of calls made to API packages.

Note that it is more or less impossible to imagine all the types of attacks that a savvy attacker may come up with - hence, in this paper, we limit our claims of robustness to these types of attacks.

a) Fake API package call feature attack: Here, attackers try to evade FARM by increasing the percentage of fake API package calls made, i.e. by adding more and more fake API package calls into the code. Table VI shows that the impact of this attack on FARM is just 10.47-72.12% than the impact on the baselines — on average, across the 12 classification problems, the impact on FARM is 36%, i.e. FARM is about 3 times as robust as the baselines across the 12 problems studied in this paper. Figures 5 and 6 show the impact of this attack on the best version of FARM (blue line with square markers) compared to the best baseline (yellow line with dot markers) as the percentage of fake calls increases in the rooting app vs. goodware and rooting app vs. other malware classification problems respectively.

Surprisingly, as more fake API package calls are made, it becomes easier for classifiers to identify rooting malware. This suggests that the malicious behavior of malware is related to the API package calls that they make. For example, Android Banking Trojans call API *android.app.admin* to hijack a smartphone's administrative features at the system level, while it is not commonly called in Android Goodware. Thus, when we simulate the attacker's behavior and increasing the Fake Call Percentage in malware, the performance of the classifier using both our best setting and baseline improves because the fake calls may involve the malware calls many more API calls than a piece of goodware would ordinarily make. FARM always achieve better F1 performance, especially when the attacker injects only a small percentage of fake API package calls (which is the best strategy for him as this is when both FARM and the baselines' predictive accuracy is lowest in this situation).

b) Fake permission attack: Second, we assume that attackers try to evade malware detection by increasing the number of permissions they seek. Table VII shows that on average, the impact of this attack on FARMis 10.25-74.42% of the impact on the best baseline, with the average impact on FARM being 35.74%. Thus, as in the case of the first attack, FARM is about 3 times as robust to this attack than the best baseline.

The "fake permission" percentage in Figures 7 and 8 refer to the percentage of requested permissions that are fake. The figures respectively show the results of distinguishing between rooting malware and goodware on the one hand, and other malware on the other hand. We see that as more permissions are required, both FARM and the baselines do a better job in detecting rooting malware. But again, the best case scenario for the attacker is when the percentage of fake (unused) permissions is below about 8% and in this case, FARM beats the baselines. FARM performs better than the best baseline because malware also achieves its malicious function by calling system permissions in the manifest file. For example, Android Spyware uses system permissions *permission:RECEIVE_SMS* and *permission:READ_SMS* to steal messages from the smartphone while common Android Goodware does not. Also, both common

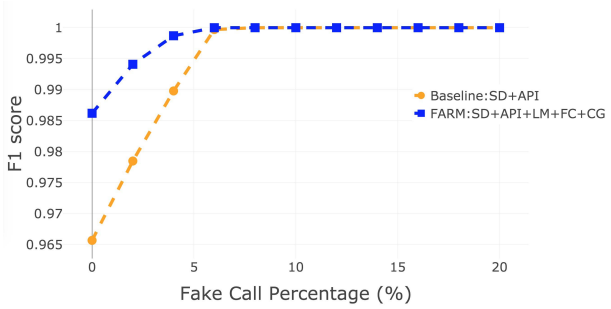

Fig. 18. Impact of fake API package call attack on Android Spyware detection: Other Malware vs. Spyware (No-Isomorphic).

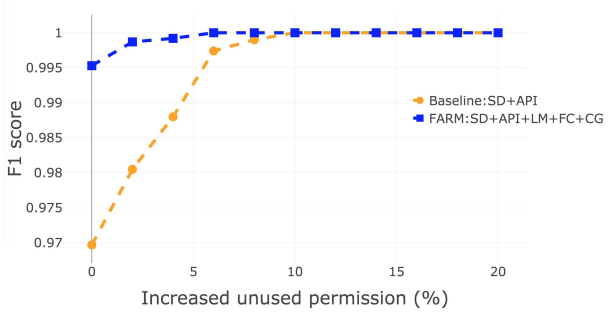

Fig. 19. Impact of increased percentage of permissions attack on Android Spyware detection: Goodware vs. Spyware (No-Isomorphic).

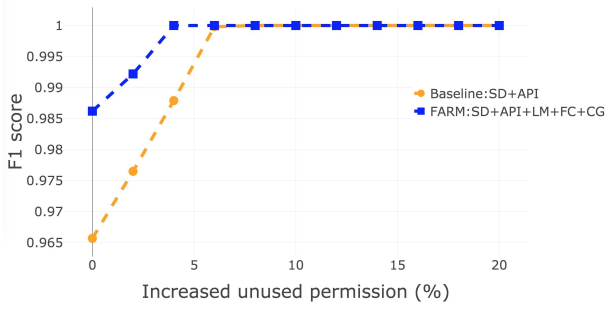

Fig. 20. Impact of increased percentage of permissions attack on Android Spyware detection: Other Malware vs. Spyware (No-Isomorphic).

Android Goodware and Android Spyware do not call the permission *android.permission.SET_TIME*, which allows the application to set the system time. When the attacker behavior increases the Unused Permissions Percentage, it the classifiers' job becomes easier to distinguish the adapted malware if it calls the permission *android.permission.SET_TIME*. The performance of both classifiers increases at the same time.

c) Reduced API feature attack: Third, we assume that attackers are more strategic and capable — we allow them to selectively drop some API package calls by 1 when the original value is at least 2. Table VIII shows that the impact of this attack on FARM ranges from −3.64-6.14, suggesting a wide variation. On 11 of 12 cases, FARM outperforms the best baseline, but in one case (Other malware vs. Rooting malware), the best baseline outperforms FARM. Again, on average, FARM performs very well, with the accuracy of FARM often improving under this attack. This is because the modified malware achieves its malicious purpose by calling specific API calls and because the number of called APIs can be decreased but they cannot be fully removed. The results on Rooting

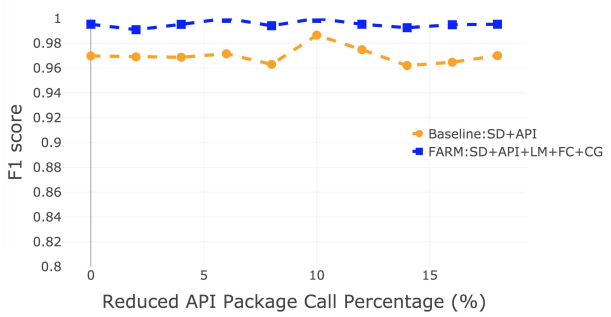

Fig. 21. Impact of reduced percentage of API package call attack on Android Spyware detection: Goodware vs. Spyware (No-Isomorphic).

malware detection are shown in Figures 9 and 10. Unlike the results from the previous two kinds of attack, we see that the reduced API feature attack is harder for both FARM and traditional classifiers to adapt to. However, the situation is worse for the baseline classifiers. When distinguishing between rooting malware and goodware, the F1 performance goes down slightly as the number of API package calls is reduced. The reason might be that rooting malware gets less malicious and more similar to goodware in this case. However, the F1 score goes up when distinguishing rooting malware from other malware. The reason might be that as rooting malware is getting more similar to goodware, it ends up being more distinct from other malware. Table VI, Table VII and Table VIII show 3 kinds of applied attack during the robustness test, and impact score is calculated according to the average of 0% to 20% increased or decreased number of APIs or permissions. The performance under attack is always increasing because our classifiers distinguish Goodware vs. Malware on API or Permission features. When we simulate the attack, we increase the number of unused APIs or permissions in malware, and so can detect the malware easier because some API or permission features may not be used by both goodware or malware, but now more malware calls the common unused feature, leading to better classification results. Again, FARM performs better than the baseline. When we decrease 1 for some of called APIs (frequency $\geq$ 2 to keep its malicious function) in malware, no obvious change on the performance because the feature space doesn't change too much compared to the previous two attacks. At the same time, FARM still has better performance.

C. Discovery of New Rooting Malware

FARM has successfully labeled two malware samples on VirusTotal[12] as rooting malware before this was observed by any of the 61 anti-virus engines on VirusTotal. Moreover, on a phone running Android version *4.4.4*, these are labeled as goodware as Figure 24 shows. We reported these two samples to Google's Android Security Team who have confirmed the findings.

The first malware has the (common) name "App Market"[13] and disguises itself as a normal third party application market

[12]https://www.virustotal.com

[13]SHA256: *1ff2c23d3e6558ad4394ac3eb339c1bc4952eecfa45a35e3eb0e206db8568925*

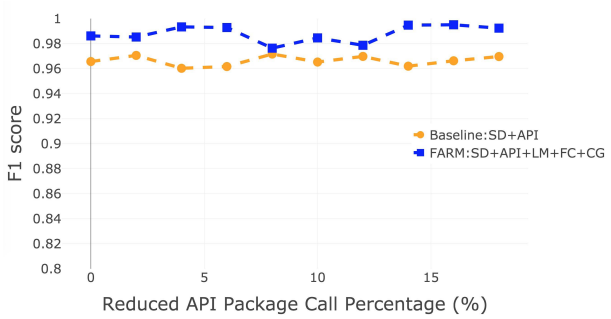

Fig. 22. Impact of reduced percentage of API package call attack on Android Spyware detection: Other Malware vs. Spyware (No-Isomorphic).

TABLE VI
AVERAGE IMPACT SCORE a OF FARM OVER THE BEST BASELINE ON INCREASED FAKE API PACKAGE CALL ATTACK

Classification Problem	a
Goodware vs. Rooting (No-Isomorphic)	0.1846
Goodware vs. Rooting (Isomorphic)	0.6824
Goodware vs. Banking Trojan (No-Isomorphic)	0.1047
Goodware vs. Banking Trojan (Isomorphic)	0.7212
Goodware vs. Spyware (No-Isomorphic)	0.1138
Goodware vs. Spyware (Isomorphic)	0.4505
Other-malware vs. Rooting (No-Isomorphic)	0.2310
Other-malware vs. Rooting (Isomorphic)	0.2989
Other-malware vs. Banking Trojan (No-Isomorphic)	0.1728
Other-malware vs. Banking Trojan (Isomorphic)	0.5348
Other-malware vs. Spyware (No-Isomorphic)	0.1047
Other-malware vs. Spyware (Isomorphic)	0.7212

TABLE VII
AVERAGE IMPACT SCORE a OF FARM OVER THE BEST BASELINE ON INCREASED PERCENTAGE OF PERMISSIONS ATTACK

Classification Problem	a
Goodware vs. Rooting (No-Isomorphic)	0.1952
Goodware vs. Rooting (Isomorphic)	0.6063
Goodware vs. Banking Trojan (No-Isomorphic)	0.1025
Goodware vs. Banking Trojan (Isomorphic)	0.7442
Goodware vs. Spyware (No-Isomorphic)	0.1376
Goodware vs. Spyware (Isomorphic)	0.4623
Other-malware vs. Rooting (No-Isomorphic)	0.2286
Other-malware vs. Rooting (Isomorphic)	0.2617
Other-malware vs. Banking Trojan (No-Isomorphic)	0.1791
Other-malware vs. Banking Trojan (Isomorphic)	0.5250
Other-malware vs. Spyware (No-Isomorphic)	0.1025
Other-malware vs. Spyware (Isomorphic)	0.7442

APK. When the user installs this APK, it asks for 14 permissions in total, including some dangerous permissions such as *WRITE_CALENDAR* and *WRITE_EXTERNAL_STORAGE*.[14] After installation, a number of malicious behaviors end up occurring: it keeps asking for new permissions, automatically downloads new apps, and bypasses the lock screen.

The second malware with the name "MoboMarket"[15] is actually impersonating a benign application also called MoboMarket and asks for dangerous permissions such as *WRITE_EXTERNAL_STORAGE*. After digging into its

[14]https://developer.android.com/guide/topics/permissions/overview

[15]SHA256: *886238c0d4894bd346cd7c3c5585d9c48b50d1ba73c90284f33eed1c0a5336df*

TABLE VIII
AVERAGE IMPACT SCORE a OF FARM OVER THE BEST BASELINE ON REDUCED PERCENTAGE OF API PACKAGE CALL ATTACK

Classification Problem	a
Goodware vs. Rooting (No-Isomorphic)	-0.7319
Goodware vs. Rooting (Isomorphic)	-3.2924
Goodware vs. Banking Trojan (No-Isomorphic)	-1.1666
Goodware vs. Banking Trojan (Isomorphic)	0.6750
Goodware vs. Spyware (No-Isomorphic)	-3.6451
Goodware vs. Spyware (Isomorphic)	-0.9673
Other-malware vs. Rooting (No-Isomorphic)	0.2623
Other-malware vs. Rooting (Isomorphic)	6.1428
Other-malware vs. Banking Trojan (No-Isomorphic)	-0.0247
Other-malware vs. Banking Trojan (Isomorphic)	-0.0212
Other-malware vs. Spyware (No-Isomorphic)	-1.1666
Other-malware vs. Spyware (Isomorphic)	0.6750

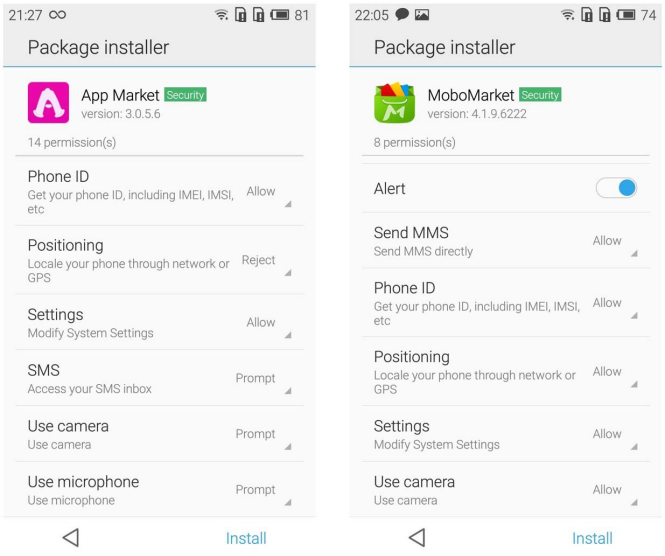

Fig. 23. Two newly detected rooting malware evade default Android Security check.

obfuscated java source code, we found some code snippets which are directly related to rooting behavior as seen in Figure 24. We found that its source code includes a public class *RequestRootActivity.java* that asks for root privilege on the infected device. We also discovered out that this "MoboMarket" is actually a malware variant of the authentic "MoboMarket" application, which has rooting functionality. Again, the Google Android Security Team confirms that this newly labeled rooting malware impersonates a popular rooting APK called KingRoot.

VII. LIMITATIONS OF THE FARM APPROACH

The FARM approach has three limitations that we discuss briefly below.

1) Though FARM's feature transformations can be applied to any set of "base" features, it is important that this set of base features be selected judiciously and be capable of making good predictions. In this paper, we chose base features that have been shown in the literature to be useful for classifying Android apps into benign vs. malicious samples.

```
 27: public class RequestRootActivity extends Activity {
 33:     /* renamed from: com.baidu.androidstore.ui.RequestRootActivity$1 */
 36:         final /* synthetic */ RequestRootActivity f8855a;
 38:         C24901(RequestRootActivity requestRootActivity) {
 39:             this.f8855a = requestRootActivity;
 52:     /* renamed from: com.baidu.androidstore.ui.RequestRootActivity$2 */
 55:         final /* synthetic */ RequestRootActivity f8856a;
 57:         C24912(RequestRootActivity requestRootActivity) {
 58:             this.f8856a = requestRootActivity;
 72:     /* renamed from: com.baidu.androidstore.ui.RequestRootActivity$3 */
 75:         final /* synthetic */ RequestRootActivity f8857a;
 77:         C24923(RequestRootActivity requestRootActivity) {
 78:             this.f8857a = requestRootActivity;
 87:     /* renamed from: com.baidu.androidstore.ui.RequestRootActivity$4 */
 90:         final /* synthetic */ RequestRootActivity f8858a;
 92:         C24934(RequestRootActivity requestRootActivity) {
 93:             this.f8858a = requestRootActivity;
211:         auVar.m15378b((int) R.string.dialog_request_root_message);
```

Fig. 24. Malicious source code snippet from malware MoboMarket.

2) As in much of machine learning research both inside and outside cybersecurity, there is a critical need to find the values of the hyperparameter settings that yield the best prediction results. We have adopted a grid search based method to address this problem in this paper, but an analytic solution could be helpful for future work.
3) We have shown that FARM is robust against three types of attack. However, there may be other kinds of attacks (e.g. obfuscated gradient based attacks [8] or attacks that do not depend on API function calls that we have not tested against). While we do not expect to find classifiers that are robust against every type of attack, identifying a larger space of attacks and showing how FARM either is robust to those attacks or could be modified to withstand those attacks is an important future research topic. While FARM's feature transformations are defined even if completely different types of features are used, their effectiveness with new or very different features remains to be explored.

VIII. Conclusion and Future Work

In this paper, we make the following contributions: (i) first, we propose three new feature transformation techniques that can be used to generate feature vectors that are very hard to reverse engineer, (ii) we propose the FARM techniques that use these transforms to predict whether a given Android APK is a form of malware or not — we consider three forms of malware, namely spyware, banking trojans and rooting malware. (iii) we propose three new kinds of attacks that a malicious hacker might take to evade standard classifiers and show that FARM is quite robust against these kinds of attacks. In particular, when there are no attacks, FARM slightly outperforms various baselines and when these three attacks are used, FARM is on average about 3 times more robust than the baselines. Finally, our work is not purely theoretical: FARM has discovered two Android APKs to be rooting apps before any of the 61 anti-viruses on VirusTotal came to the same conclusion. These samples were reported to Google's Android Security Team who have confirmed the labeling of these samples as rooting apps.

References

[1] (2011). *Number of the Week: At Least 34% of Android Malware is Stealing Your Data*. Accessed: May 20, 2019. [Online]. Available: https://www.kaspersky.com/about/press-releases/2011_number-of-the-week- at-least-34-of-android-malware-is-stealing-your-data

[2] (2012). *2011 Mobile Threats Report*. Accessed: May 20, 2019. [Online]. Available: https://www.juniper.net/us/en/local/pdf/additional-resources/jnpr-2011-mobile-threats-report.pdf

[3] (2019). *Android Security & Privacy 2018 Year in Review*. Accessed: May 20, 2019. [Online]. Available: https://source.android.com/security/reports/Google_Android_Security_2018_Report_Final.pdf

[4] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining API-level features for robust malware detection in android," in *Security and Privacy in Communication Networks*. Cham, Switzerland: Springer, 2013.

[5] M. Almeida *et al.*, "CHIMP: Crowdsourcing human inputs for mobile phones," in *Proc. World Wide Web Conf.*, 2018, pp. 45–54.

[6] I. Amaya, J. C. Ortiz-Bayliss, A. E. Gutierrez-Rodriguez, H. Terashima-Marin, and C. A. C. Coello, "Improving hyper-heuristic performance through feature transformation," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2017, pp. 2614–2621.

[7] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *Proc. NDSS*, 2014, pp. 23–26.

[8] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," 2018, *arXiv:1802.00420*. [Online]. Available: http://arxiv.org/abs/1802.00420

[9] C. Bai, Q. Han, G. Mezzour, F. Pierazzi, and V. S. Subrahmanian, "DBank: Predictive behavioral analysis of recent Android banking trojans," *IEEE Trans. Dependable Secure Comput.*, to be published.

[10] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, and A.-R. Sadeghi, "Xmandroid: A new Android evolution to mitigate privilege escalation attacks," Technische Univ. Darmstadt, Darmstadt, Germany, Tech. Rep. TR-2011-04, 2011.

[11] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: Behavior-based malware detection system for Android," in *Proc. SPSM*, 2011, pp. 15–26.

[12] L. Cen, C. S. Gates, L. Si, and N. Li, "A probabilistic discriminative model for Android malware detection with decompiled source code," *IEEE Trans. Dependable Secure Comput.*, vol. 12, no. 4, pp. 400–412, Jul. 2015.

[13] T. Chakraborty, F. Pierazzi, and V. S. Subrahmanian, "EC2: Ensemble clustering and classification for predicting Android malware families," *IEEE Trans. Dependable Secure Comput.*, to be published.

[14] Z. Chen, K. Wang, X. Wang, P. Peng, E. Izquierdo, and L. Lin, "Deep co-space: Sample mining across feature transformation for semi-supervised learning," *IEEE Trans. Circuits Syst. for Video Technol.*, vol. 28, no. 10, pp. 2667–2678, Oct. 2018.

[15] I. Gasparis, Z. Qian, C. Song, and S. V. Krishnamurthy, "Detecting Android root exploits by learning from root providers," in *Proc. 26th USENIX Secur. Symp.*, 2017, pp. 1129–1144.

[16] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial examples for malware detection," in *Proc. Eur. Symp. Res. Comput. Secur.* Cham, Switzerland: Springer, 2017, pp. 62–79.

[17] S. Hou, Y. Ye, Y. Song, and M. Abdulhayoglu, "Hindroid: An intelligent Android malware detection system based on structured heterogeneous information network," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2017, pp. 1507–1515.

[18] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on GAN," 2017, *arXiv:1702.05983*. [Online]. Available: http://arxiv.org/abs/1702.05983

[19] X. Huang, L. Wu, E. Chen, H. Zhu, Q. Liu, and Y. Wang, "Incremental matrix factorization: A linear feature transformation perspective," in *Proc. 26th Int. Joint Conf. Artif. Intell.*, Aug. 2017, pp. 1901–1908.

[20] U. Khurana, D. Turaga, H. Samulowitz, and S. Parthasrathy, "Cognito: Automated feature engineering for supervised learning," in *Proc. IEEE 16th Int. Conf. Data Mining Workshops (ICDMW)*, Dec. 2016, pp. 1304–1307.

[21] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an, and H. Ye, "Significant permission identification for Machine-Learning-Based Android malware detection," *IEEE Trans Ind. Informat.*, vol. 14, no. 7, pp. 3216–3225, Jul. 2018.

[22] Y. Li, J. Jang, X. Hu, and X. Ou, "Android malware clustering through malicious payload mining," in *Proc. Int. Symp. Res. Attacks, Intrusions, Defenses*. Cham, Switzerland: Springer, 2017, pp. 192–214.

[23] M. Lindorfer, M. Neugschwandtner, and C. Platzer, "MARVIN: Efficient and comprehensive mobile app classification through static and dynamic analysis," in *Proc. IEEE 39th Annu. Comput. Softw. Appl. Conf.*, Jul. 2015, pp. 422–433.

[24] I. Lunden. (2015). *6.1B Smartphone Users Globally by 2020, Overtaking Basic Fixed Phone Subscriptions*. Accessed: May 21, 2019. [Online]. Available: http://techcrunch.com/2015/06/02/6-1b-smartphone-users-globally-by-2020-overtaking-basic-fixed-phone-subscriptions
[25] W. Ma, P. Duan, S. Liu, G. Gu, and J.-C. Liu, "Shadow attacks: Automatically evading system-call-behavior based malware detection," *J. Comput. Virol.*, vol. 8, nos. 1–2, pp. 1–13, Dec. 2011.
[26] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "MaMaDroid: Detecting Android malware by building Markov chains of behavioral models," in *Proc. NDSS*, 2017, pp. 1–34.
[27] N. McLaughlin *et al.*, "Deep Android malware detection," in *Proc. 7th ACM Conf. Data Appl. Secur. Privacy*, 2017, pp. 301–308.
[28] L. Onwuzurike, M. Almeida, E. Mariconti, J. Blackburn, G. Stringhini, and E. De Cristofaro, "A family of droids-Android malware detection via behavioral modeling: Static vs dynamic analysis," in *Proc. 16th Annu. Conf. Privacy, Secur. Trust (PST)*, Aug. 2018, pp. 1–10.
[29] M. Pechenizkiy, A. Tsymbal, and S. Puuronen, "PCA-based feature transformation for classification: Issues in medical diagnostics," in *Proc. 17th IEEE Symp. Computer-Based Med. Syst.*, Jun. 2004, pp. 535–540.
[30] V. Rastogi, Y. Chen, and X. Jiang, "Droidchameleon: Evaluating Android anti-malware against transformation attacks," in *Proc. 8th ACM SIGSAC Symp. Inf., Comput. Commun. Secur.*, 2013, pp. 329–334.
[31] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "MADAM: Effective and efficient behavior-based Android malware detection and prevention," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 1, pp. 83–97, Jan. 2018.
[32] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, "CopperDroid: Automatic reconstruction of Android malware behaviors," in *Proc. NDSS*, 2015, pp. 1–15.
[33] K. Tian, D. Yao, B. G. Ryder, G. Tan, and G. Peng, "Detection of repackaged Android malware with code-heterogeneity features," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 1, pp. 64–77, Jan. 2020.
[34] R. Unuchek. (Jul. 2019). *Rooting Your Android: Advantages, Disadvantages, and Snags*. [Online]. Available: https://www.kaspersky.com/blog/android-root-faq/17135/
[35] 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 Secur.*, vol. 9, no. 11, pp. 1869–1882, Nov. 2014.
[36] G. Wu and E. Y. Chang, "Adaptive feature-space conformal transformation for imbalanced-data learning," in *Proc. 20th Int. Conf. Mach. Learn. (ICML)*, 2003, pp. 816–823.
[37] X. Wu, W. Zuo, L. Lin, W. Jia, and D. Zhang, "F-SVM: Combination of feature transformation and SVM learning via convex relaxation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 11, pp. 5185–5199, Nov. 2018.
[38] L. Xing, X. Pan, R. Wang, K. Yuan, and X. Wang, "Upgrading your android, elevating my malware: Privilege escalation through mobile OS updating," in *Proc. IEEE Symp. Secur. Privacy*, May 2014, pp. 393–408.
[39] L. K. Yan and H. Yin, "Droidscope: Seamlessly reconstructing the OS and dalvik semantic views for dynamic Android malware analysis," in *Proc. 21st USENIX Secur. Symp.*, 2012, pp. 569–584.
[40] H. Zhang, D. She, and Z. Qian, "Android root and its providers: A double-edged sword," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 1093–1104.
[41] M. Zheng, M. Sun, and J. C. S. Lui, "DroidTrace: A ptrace based Android dynamic analysis system with forward execution capability," in *Proc. Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Aug. 2014, pp. 128–133.
[42] Z. Zhu and T. Dumitraş, "Featuresmith: Automatically engineering features for malware detection by mining the security literature," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 767–778.

Qian Han received the B.Eng. degree from the Department of Electronic Engineering, Tsinghua University, in 2016. He is currently pursuing the Ph.D. degree with Dartmouth College, advised by Prof. V. S. Subrahmanian. In 2015, he spent three months as a Visiting Research Assistant at Nanyang Technological University, Singapore. His research interests lie in cybersecurity, data-mining, game theory, and social network analysis.

V. S. Subrahmanian is currently a Distinguished Professor in cybersecurity, technology, and society with Dartmouth College, and also the Director of the Institute for Security, Technology, and Society at Dartmouth. He previously served as a Professor of computer science at the University of Maryland from 1989 to 2017, where he created and headed both the Lab for Computational Cultural Dynamics and the Center for Digital International Government. He also served as the Director of the University of Maryland's Institute for Advanced Computer Studies for over six years. He is an expert on big data analytics, including methods to analyze text/geospatial/relational/social network data, learn behavioral models from the data, forecast actions, and influence behaviors with applications to cybersecurity and counterterrorism. He has written five books, edited ten, and published over 300 refereed articles. He is a fellow of the American Association for the Advancement of Science and the Association for the Advancement of Artificial Intelligence; moreover, he received numerous other honors and awards. His work has been featured in numerous outlets such as the Baltimore Sun, The Economist, Science, Nature, The Washington Post, and American Public Media. He serves on the editorial boards of numerous journals, including Science, the Board of Directors of the Development Gateway Foundation (set up by the World Bank), SentiMetrix, Inc., and on the Research Advisory Board of Tata Consultancy Services. He previously served on the DARPA's Executive Advisory Council on Advanced Logistics and as an ad-hoc member of the US Air Force Science Advisory Board.

Yanhai Xiong received the bachelor's degree in automation from the University of Science and Technological University of China, and the Ph.D. degree in computer science and engineering from Nanyang Technological University, Singapore. She has been holding a post-doctoral position at Dartmouth College since July 2018. Her research interests lie in optimization, machine learning, cybersecurity, and smart cities.