On the Effectiveness of Application Characteristics in the Automatic Classification of Malware on Smartphones

Matthew Ping, Bander Alsulami, Spiros Mancoridis
Drexel University
Department of Computer Science
College of Computing and Informatics
Philadelphia, PA
{mtp57, bma48, spiros }@drexel.edu

Abstract

The increase in smartphone usage is providing impetus to malicious actors to target these devices via malware injection. This can be seen in the increasing number of malware identified in the past few years. Android, being the most commonly used platform and one that provides an open architecture, makes it the most common target for malware developers. One possible method to identify malicious code is to use the characteristics of an application such as permissions to identify an application's disposition. This paper describes a method for using application characteristics to classify sample applications as either benign or malware. Both binary and familial classification of malware samples is performed to determine whether each sample is malware or not (i.e., binary classification) and what is the familial provenance of the malware sample (i.e. familial classification). The results compare the effectiveness of the familial classification of three different commercial anti-virus engine taxonomies.

1. Introduction

The International Data Corporation (IDC) states that 341.5 million smartphones have been shipped worldwide during Q2 2015. [1] Of the various smartphone platforms, Android is by far the largest, holding approximately 82.8% of the smartphone market worldwide. [1]

As mobile devices increase in popularity, additional functions and capabilities are being added to the newest generation of smartphones. Online shopping, banking, and health are just a few of the new capabilities that smartphones have enabled. While these new capabilities provide benefits to users, they also represent an impetus to malware developers seeking to exploit smartphone users. Malware developers may seek financial gain or may seek to simply disrupt the standard operations of the device. [11, 26]

In 2015, Kaspersky Lab detected 884,774 new mobile malware samples. This represents a three-fold increase from 2014, when they detected 295,539 mobile malware samples. [3] Further examining malware specifically on Android, a separate report from F-Secure Labs [18] shows that in 2013, 804 new malware families or 97% of all new malware families detected were on Android. These numbers represent a real threat to the Android platform.

In order to address the issue of mobile malware, researchers have been developing novel ways to detect and classify malware. Some methods of analysis make use of system calls, [8] API calls, [23, 24] or other aspects of the OS and installed apps such as permissions. [4, 21, 20, 22, 16] Tracking system and/or API calls typically relies on dynamic analysis of the executing malware, while permission analysis focuses on static analysis of the app. Both approaches are complementary in nature. Dynamic analysis techniques can reveal complex issues that can only be detected during execution, while examining application characteristics can provide a static level of insight before executing the application.

This paper describes the use of application characteristics, specifically Android permissions, to perform binary and familial classification of malware. Analysis of permissions can be done using a static approach or dynamically at run time, which makes permissions a versatile feature. Many of the permissions-based approaches that are noted above only perform a binary classification of malware. Instead of only employing a binary classification approach, our work examines both binary and familial classification. Furthermore, our technique compares three different familial taxonomies that are used in three commercial AV products: Microsoft Security Essentials, Kaspersky Labs, and ESET-NOD32. The accuracy of classification for each tax-

onomy is also discussed and compared. Because this paper focuses on the accuracy of our technique rather than the accuracy of commercial anti-virus (AV) scanners, only samples that are identified by all three AV engines are considered. While this reduces the available number of apps, since not all samples are detected by all three engines, it does provide ground truth when comparing the results against the AV engine taxonomies. Additionally, this approach improves on previous work by using analysis techniques to classify the malware into families instead of only performing binary classification (benign or malicious) of malware. The results show that this approach is capable of classifying families at a rate similar to various binary classification approaches that limit themselves to yes (is malware) or no (is not malware) answers.

The remainder of this paper is organized as follows: Section 2 describes related work that is relevant to this research. Section 3 describes the Android platform and the permissions system used by Android. The experimental setup, organization of the data being used for the experiment, and the algorithms used in our analysis are discussed in Section 4. The results of the experiment are discussed in Section 5. Potential future work is outlined in Section 6 and our conclusions are summarized in Section 7.

2. Related Work

While this paper focuses on analyzing application characteristics, discussion of existing dynamic analysis approaches can provide insight to alternative methods of malware detection and classification. One dynamic approach is Crowdroid [8], which traces system calls. An app is used by participants in the study to capture the system call information and store them in a remote database where the results are analyzed. DroidScope [24] makes use of both system level calls and API calls to detect and analyze certain malware samples. Aside from simply tracing calls, it also modified the QEMU [1] emulator to support a more detailed analysis of how a few malware samples behave while executing. Another interesting approach called DroidMat [23] makes use of API call tracing alongside of Intents [2] and Permissions. Android Intents are mechanisms for passing information between executing applications. A detailed discussion of Android's permission system is in section 3 of this paper. DroidMat combines this information for use in a k-Nearest Neighbor technique to detect malware. Dynamic approaches require access to the executing device or VM, while a static analysis can be performed on the application without executing it. Furthermore, many dynamic approaches on the Android platform require root access in order to capture information such as system calls and API calls in real-time.

In contrast, static or application characteristic analysis approaches examine applications without executing them. Research from Deshotels *et. al.* [10] describes a method for examining an Android app by first partitioning its classes into modules. These modules are then compared to a signature of a malware family to determine if they are malware or not. A more comprehensive analysis of malware using permissions is from Kang *et. al.* [17] where the researchers do both binary classification and familial classification. Using a Naive Bayes algorithm, they are able to assign a probability to a permission to determine the likelihood of it being used in malware. They also analyze the use of shell commands and other APIs to generate a "similarity score", which can then be used in familial classification of the malware using F-Secure's familial taxonomy. DroidMiner [25] makes use of dependency graphs generated from Android life cycle functions, use of any APIs that require Android permissions, and any sensitive resources that are accessed such as `content://sms/inbox/`. Weightings are used when classifying apps to determine how similar the app is to either benign applications or malware. Similarly, CHABADA [15] analyzes the application description and the sensitive APIs that are used by an app. Another approach by Avdiienko *et. al.* [5] uses Flowdroid to perform taint analysis of the data flows within an app.

A number of researchers focus on a specific set of application characteristics such as permissions in their research. For instance, Aung and Zaw [4] present a method using various machine learning techniques to analyze a dataset of Android app permissions. The dataset produces a binary classification of the permissions that an app has and identifies the app as "goodware" or "malware". They apply various feature selection methods such as Gini-impurities and entropy (Information Gain) depending on the machine learning algorithm. PUMA (Permission Usage to detect Malware in Android) [21] also makes use of various machine learning algorithms. However, PUMA focuses on different algorithms and also discusses the permissions that a typical malware has along with the number of permissions a typical malware has.

While these papers focus on using only Android permissions as a method for malware detection, others incorporate additional information. For instance, Pehlvan *et. al.* [20] make use of version name and version code in addition to permissions. Their results demonstrate an improvement over the approaches that only use permissions. Similarly, APK Auditor [22] also makes use of app name and version information in their detection method. However, rather than using traditional machine learning techniques to analyze applications, they instead determine how often a permission is used by malware in relation to all applications. This allows

[1] `http://www.qemu.org/`

[2] `https://developer.android.com/guide/components/intents-filters.html`

APK Auditor to "score" an Android app to give a confidence level on it being benign or malware. Another approach by Idrees and Rajarajan [16] incorporates Android Intents in addition to permissions. Android Intents are a mechanism for communication within an app or between various apps that can potentially be used to trigger malicious or potentially unwanted actions. They make use of machine learning techniques such as Naive Bayes to classify permissions and intents as either "normal" or "dangerous". In MAST [9], Chakradeo *et. al.* propose a lightweight system using permissions, intent filters, the presence of native code, and the presence of zip files. The features are run through a series of selection and merging algorithms to generate a ranking of the risk associated with this application. Gomez *et. al.* [13] analyze four different malware variants in their paper. They examine the permissions required by those malware to determine commonalities. However, their work only examine four samples: DroidDream, DroidDreamLight, Zsone SMS, and Geinimi. WHYPER, one of the more notable contributions in this space, uses the application description, app API documentation, and permission information to support binary classification. [19]

A number of these approaches use permissions and potentially other data points to detect if an app is benign or malware. Even when classification of malware is done, a limited number of malware families are considered. While all of these approaches are novel and effective, additional work is still possible in this space. The technique this paper describes seeks to build on the efforts discussed above. Permission based detection has shown issues previously with over-privileged benign apps and similar malware families. This technique shows similar accuracy, higher precision, higher recall, and higher F-Score while limiting those problems. Using three different commercial AV malware family taxonomies in this technique shows how it can be applied across a number of taxonomies. Both binary and familial classification is done to compare the effectiveness of this technique against other similar works. The remainder of this paper discusses the setup and evaluation of the technique.

3. Overview of Android's Permission System

The Android OS is an open source platform that is commonly used for, but is not limited to, mobile devices. Android has the ability to run apps coded in Java by using a virtual machine called the Dalvik Virtual Machine (DVM). The DVM is a virtual machine created for the Android platform that executes compiled bytecode. Some functions of the device are protected through a permissions scheme. These permissions include providing access to device functions such as "`INTERNET`", "`BOOT_COMPLETED`", and "`SMS_RECEIVED`." In order to access functions such as these an app has to declare permissions it requires or it will not be able to run. These permissions are defined in a file called *AndroidManifest.xml*, which is part of the app. Upon attempting to install the application on Android 5.0 (Lollipop) or an earlier version, a user is prompted to agree to all declared permissions or the app will not install. Starting with Android 6.0, the permission system is different. The same permissions are used; however, permissions are "grouped" together to allow users to grant or reject an individual application from using all permissions in that group. [2] Further discussion of the impact of this change can be found in the Future Work section of this paper.

Android malware is typically an app and, therefore, also must declare permissions required to run. Research shows that some permissions are used more frequently by malware. For instance, Zhuo and Jiang [27] dissect a number of malware samples and show that "`BOOT_COMPLETED`" and "`SMS_RECEIVED`" are among the most utilized permissions by malware. Some malware such as `Gemini` make use of the "`BOOT_COMPLETED`" permission to bootstrap adware once the boot process has completed. Another example, is the `zSone` malware, which makes use of the "`SMS_RECEIVED`" permission. It intercepts texts from "10086" and "10010", to prevent the user from seeing texts coming from those premium numbers.

4. Experimental Setup

This section describes the experiment in terms of how the dataset is built, a description of the classification algorithms, and how the classification tests are structured.

4.1. Dataset Construction

In order to measure the effectiveness of this method, the dataset includes both benign samples and malware samples. Benign samples are from the Google Play Store, which is the official app marketplace for Android apps. An Unofficial Google Play API library[3] is used to download the benign samples. There are 539 unique samples that were downloaded from the store between 2 June 2015 and 3 June 2015. These samples include the top 20 free apps of each category in the Google Play Store.

Malware samples are collected from a torrent available on VirusShare, a website that provides information about how to obtain the latest malware samples. The malware dataset comes from a torrent[4] that contains APKs from 6 May 2013 to 24 March 2014. In order to obtain ground truth labeling of malware family taxonomies, VirusTotal, which

[3] `https://github.com/egirault/googleplay-api`

[4] `http://tracker.virusshare.com:6969/torrents/VirusShare_Android_20140324.zip.torrent?1FF75484B32741141659DC239611D20F5500B780`

Microsoft	BaseBridge, BoxerSm, BoxerSms, CVE-2011-182, DDLight, DroidKrungF, DroidKrungFu, FakeInstSms, FakePlaye, Geinimi, Kmin, Legana, Lotoo, Pjapps, Plankton, Yzhc
Kaspersky	Adrd, BaseBrid, DorDrae, FakeInst, FakePlayer, Geinimi, Kmin, KungFu, Lotoor, Meds, Mseg, Opfake, Plangton, Stealer, Yzhc
ESET-NOD32	AB, AdsWo, BaseBridge, Boxer, Domob, DroidKungFu, DroidKungFu/M, DroidKungFu/W, DroidRooter, FakeInst, FakePlayer, Geinimi, Kmin, Lightdd, Lotoor, MailCrypt, Mseg, PJApps, Plankton, RageCage, Stealer, Wooboo, YZHC

Table 1: Family names of the malware samples that are used in this research organized by AV engine.

is an online database containing the scan results from AV engines, is used. The results for Microsoft, Kaspersky, and ESET-NOD32 are used when labeling the malware datasets.

Table 1 shows the malware samples that are part of the dataset along with the family names from each AV engine. The table is a simple listing in alphabetical order of the malware family names that are used in this research. Each AV engine assigns a malicious app to a malware family. Family classifications are not completely consistent between the AV engines, which can cause difficulty when attempting to classify malware. Consider a more popular malware commonly referred to as `DroidKungFu`. Microsoft refers to this as both `DroidKrungF` and `DroidKrungFu`, while Kaspersky calls it `KungFu` and ESET-NOD32 differentiates between a base family name and two variants (`M` and `W`). The impact this type of specialization has on the results is described in more detail in Section 5. While differences exist, the three taxonomies largely agree on the family classification of the selected malware samples.

Both benign and malware APKs have *AndroidManifest.xml* files that define the permissions that an app is requesting. In Android 5.0 (Lollipop), and earlier, a user installing an app automatically grants the permissions the app is requesting. Without specific permissions, apps are not capable of accessing certain device functions. In order to extract what permissions an app is requesting, the Android Application Packaging Tool (`AAPT`) is used to extract the "badging" of the app. Badging is a representation of information from the *AndroidManifest.xml* file, which contains permission information. Alternatively, the `requestedPermissions` attribute of the `PackageInfo` class can also identify the permissions of an app. [14] Collecting permissions while on a device can allow for live analysis of the application, aiding in dynamic analysis or hybrid analysis methods. This versatility makes permissions a useful feature in classifying malware.

Assembly of the data set involves collecting both a binary representation of the requested permissions and identifying information about the app. The 152 permissions that are commonly present in Android are represented as a single bit in a row of data. If the bit is set to 0, the permission is not requested by the app and if it is set to 1, the permission is requested by the app. Identifying information is defined as either the malware family name or an indication that the sample is benign. Benign sample labels are "-1", while malware sample labels are strings representing the malware family name as found in one of the AV engines. Some examples of malware family names from the AV engines are: `TrojanSMS`, `AdDisplay`, `Kmin`, *etc*. Example rows of benign and malware data are shown below:

Benign - [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, -1]

Malware - [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 'Kmin']

For each AV engine, the malware data are labeled with the family name from that specific engine. The data that are labeled for a specific engine are combined with the benign samples that are also labeled. This process results in three separate datasets, one for each AV engine. Because the same malware and benign samples are used, each dataset contains 864 total samples. Holdout cross validation with randomization is then applied to the dataset, where 80% of the data is used for training and 20% is used for testing. This process allows for the experimental classification methods to be tested against three separate AV engines to simulate the efficiency of the proposed method against samples previously not seen. After assembling the datasets and introducing holdout cross validation, the datasets are run through the algorithms described in section 4.2.

4.2. Classification Algorithms

This section discusses the algorithms that are being used in the analysis of this data. The selection of these specific algorithms is reflective of certain benefits they have given the way the data is structured. As shown in the previous section, the permissions are represented as binary data and the labels are defined as text. This structure lends itself well to rule based classifiers because the goal is to determine what set of permissions can identify a specific family of malware. Additionally, the use of a nearest neighbor algorithm provides the ability to classify a row based on its distance from other identified classes. Both rule based and neighbor based classifiers are non-parametric in nature. Since the number of malware families may not be known *a priori*, using a non-parametric model allows for a more flexible interpretation of the data. In addition to classification algorithms, dimensionality reduction is employed to identify variance within the permissions. Given that the use of each permission will vary between different applications and malware families, Principal Component Analysis (PCA) can assist in understanding how much each permission varies. This information can then be used to keep the most significant vectors for use in classification. The remainder of this section describes the specific algorithms being used.

4.2.1 Tree-based Classifiers

A Decision Tree classifier generates a tree structure where the data at each node (d) is represented by N. Let the binary representation of the permissions be x and the labels (-1 for benign software and a string for malware) be y. Data may be partitioned into left and right branches at this level based on a specific instance of app permissions (p) and threshold (t).

$$N_{left}(v) = (x, y)|x_p <= t$$
$$N_{right}(v) = N \setminus N_{left}(v)$$

The model can be further refined through the use of an impurity function. Impurity functions are used to aide in deciding which branches to prune from a tree. The impurity function for a tree is:

$$I(N, v) = \frac{n_{left}}{N_m} H(N_{left}(v)) + \frac{n_{right}}{N_m} H(N_{right}(v))$$

For this paper, two possible impurity functions are being used: Gini-impurity and Information Gain. A Gini-impurity is the frequency with which a random element might be mislabeled if it was labeled randomly based on the set of possible labels. Information Gain (or Entropy) in the context of a Decision Tree is a method for defining which data are most useful for making a decision at each branch of the tree. Gini-impurity and Entropy are represented by the following formulas:

$$\text{Gini - } H(Q_m) = \sum_i p_{mi}(1 - p_{mi})$$

$$\text{Entropy - } H(Q_m) = \sum_i p_{mi} \log(p_{mi})$$

A Random Forest Classifier [7] alters the premise of Decision Trees by sampling data at random and with replacement from the training set. Depending on the size of the dimensions D, a number (d) that is less than D is selected and d variables are randomly selected. The best split from the randomly selected d variables is used to split at each node. During this process the number d remains constant, while the variables that are selected may change. As with the Decision Tree implementation, Gini-impurity and Entropy are used for pruning the tree.

The Extra Trees Classifier [12] builds on the Random Forest algorithm. Instead of simply randomizing the number of variables that are selected, this process also randomizes the split thresholds (t). In addition to randomizing the splitting threshold, extra trees also makes use of ensemble averaging. These two techniques are shown to perform better in reducing variance versus other methods. This method also makes use of Gini-impurity and Entropy to prune the tree.

4.2.2 Nearest Neighbor Classification

This work makes use of a KD tree algorithm for determining into which class a sample should be labeled. This implementation of nearest neighbor [6, p. 125] does not require the number of classes to be defined. While the number of malware families for each AV engine is known, the goal is not to build simply for the dataset, but to create a technique that can be used more broadly. In this case, the goal is to take the binary representation of permissions (x) and the labels of benign apps or malware apps (C_k) and use them to classify a new representation of permissions based. To apply this concept in a nearest neighbor algorithm, the following algorithm is used:

$$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{p(x)}$$

Given that the data are represented in a binary fashion, a KD tree implementation for Nearest Neighbor is selected because it creates a binary tree structure. The algorithm will split by x and y coordinates using vertical and horizontal lines to divide the dataset in half recursively. In the case of this technique let x be the binary representation of permissions and y be the labels. This has the advantage of constructing nearest neighbors with only $O(\log(N))$ distance computations.

Algorithm	Accuracy	Precision	Recall	F1
Decision Tree (Entropy)	91.9%	92.0%	92.1%	91.9%
Decision Tree (Gini)	94.2 %	94.2%	94.2%	94.2%
Extra Trees (Entropy)	96.5%	96.5%	96.6%	96.5%
Extra Trees (Gini)	96.0%	95.9%	96.1%	96.0%
Random Forests (Entropy)	96.5%	96.5%	96.5%	96.5%
Random Forests (Gini)	95.4%	95.4%	95.4%	95.4%
Nearest Neighbor	97.1%	97.1%	97.1%	97.1%

Table 2: Results of binary classification of malware samples.

5. Results

Results are calculated in three different ways to determine the effectiveness of this technique. First, binary classification of the samples into benign or malware is done to establish a baseline for classification. Second, familial classification of malware is done to determine the effectiveness of using permissions to identify different malware families.

To analyze the results of the classification algorithms, the following metrics are collected: Accuracy, F1-Score, Precision, and Recall. All of these metrics make use of true positive (tp), which is the number of malware samples in a class correctly labeled by the classifier; true negative (tn), which is the number of malware samples in a class that are correctly not labeled as another class; false negative (fn), which is the number of malware samples in a class incorrectly labeled by the classifier; and false positives (fp), the number of malware samples not in a class that are incorrectly classified as belonging to that class. Accuracy is the probability that a data point is correctly labeled. Precision is the probability that a data point that is retrieved is relevant. By comparison, recall is the probability that a predicted data point label is correct. F1-Score is the weighted harmonic mean of the precision and recall statistics. The respective equations for these metrics are:

$$Accuracy = \frac{tp+tn}{tp+tn+fp+fn}$$
$$Precision = \frac{tp}{tp+fp}$$
$$Recall = \frac{tp}{tp+fn}$$
$$F_1 = 2 * \frac{precision*recall}{precision+recall}$$

Table 2 shows the results of binary classification of the apps. From these results, the Nearest Neighbor algorithm is shown to be the most effective with 97.1% accuracy. Aside from the raw results, the algorithm misclassified one benign sample as malware. One of the major issues with using permissions is because of over privileged apps. In

AV Engine	Algorithm	Accuracy	Precision	Recall	F1
Microsoft	Decision Tree (Gini)	90.2 %	90.6%	92.2%	90.2%
	Extra Trees (Entropy)	92.5%	92.2%	93.6%	92.5%
	Extra Trees (Gini)	92.4%	91.7%	92.7%	92.5%
	Random Forests (Entropy)	93.1%	92.7%	94.2%	93.1%
	Random Forests (Gini)	93.1%	92.3%	93.0%	93.1%
	Nearest Neighbor	94.2%	93.9%	94.4%	94.2%
Kaspersky	Decision Tree (Gini)	91.3%	91.8%	93.0%	91.3%
	Extra Trees (Entropy)	93.6%	92.9%	93.6%	93.6%
	Extra Trees (Gini)	91.3%	91.0%	91.9%	91.3%
	Random Forests (Entropy)	94.8%	94.4%	94.5%	94.8%
	Random Forests (Gini)	94.8%	94.6%	95.0%	94.8%
	Nearest Neighbor	94.8%	94.6%	95.1%	94.8%
ESET-NOD32	Decision Tree (Gini)	85.5%	84.7%	84.8%	85.5%
	Extra Trees (Entropy)	91.3%	89.9%	90.8%	91.3%
	Extra Trees (Gini)	90.8%	89.0%	89.3%	90.8%
	Random Forests (Entropy)	90.2%	88.8%	89.7%	90.2%
	Random Forests (Gini)	89.6%	88.3%	88.8%	89.6%
	Nearest Neighbor	91.9%	90.7%	90.6%	91.9%

Table 3: Results of familial classification of malware samples by AV engine taxonomy.

this case, the app had 26 requested permissions. However, with this method, three benign apps that have 76 permissions are correctly classified. Furthermore, of apps with 15 of more permissions, all except for the previously mentioned app is successfully classified. This shows the method does well in properly classifying overprivileged apps. Conversely, malware requesting ten permissions or fewer do not perform as well. Four out of 35 malware samples are classified as benign when they are malware. Those apps have three or fewer permissions (`READ_EXTERNAL_STORAGE`, `REBOOT`, `WRITE_EXTERNAL_STORAGE`, `READ_LOGS`, `INTERNET`, `WAKE_LOCK`). This indicates that the applications are using other exploits in order to gain privileged access.

The results from familial classification in Table 3 show slightly lower accuracy than the binary classification process. Using Kaspersky's familial taxonomy in the classification algorithms result in a 94.8% accuracy with either a Random Forest algorithm or a Nearest Neighbor algorithm. Nearest Neighbor's recall is 0.1% higher than Random Forest. Classification using the familial taxonomies for Microsoft and ESET-NOD32 result in an accuracy of 94.2% and 91.9% respectively when using a Nearest Neighbor algorithm.

While this approach is largely successful, there are potential limitations. If a malware application uses the exact same permission set as a benign application, it is possible this could confuse this technique. However, this limitation would likely only occur with certain overprivileged applications using the same permissions as very specific malware exploits.

Kaspersky has the most trouble with classifying `Lotoor`, with five apps being misclassified. In one case it is misclassified as `DroidKungFu`, which is a different root exploit attack. Another misclassification is `BaseBridge`, which is a Trojan attack. The other times `Lotoor` is misclassified, it is classified as a benign application because the apps have three or fewer requested permissions. `BaseBridge`, `DroidKungFu`, and a benign make up the other four samples that are misclassified. While Microsoft has a similar accuracy rate, it also had one case where it misclassifies a variant of `BoxerSms`, which has two families for `BoxerSms`. In this case the presence of the `WAKE_LOCK` permission is the reason for the misclassification. ESET-NOD has issues with the same samples that Kaspersky and Microsoft does. Additionally, it also has problems with `AdsWo`, `Domob`, and `Wooboo` families. Among the malware that is part of this dataset, these malware families have five or fewer samples. This is the result of ESET-NOD defining 23 malware families for the selected malware apps. By comparison, Kaspersky and Microsoft define 15 and 16 families for the selected malware apps.

This demonstrates that the process is effective at familial classification over a range of samples. These results are similar to ones observed in PUMA [21] and Aung and Zaw [4], which only focuses on binary classification. It also expands on the work done by Gomez *et. al.* [13] by providing a method for classifying more malware variants than are classified in their work. The binary classification results show similar accuracy to WHYPER [19] and improve on the precision, recall, and F-Score numbers, while using fewer features. In addition to demonstrating similar or better results, the effectiveness of the approach is further demonstrated by showing results from three different malware family taxonomies.

6. Future Work

This work can be extended from a few different perspectives. First, Android Marshmallow implements a different permission scheme that provides users the ability to control the permissions to which they are agreeing. Additionally, there are "permission groups" that are defined by the system. Introducing more granular user control into the process may affect a malware's ability to operate properly. How users will interact with the permission scheme and how developers will code for the permission scheme is a potential future area of study.

Another potential area of study is the expansion of the dimensions considered in this paper. Other work cited in the Related Work section discusses the use of additional factors combined with permissions to produce a stronger result. Factors such as Android Intents and Dalvik bytecode could potentially be used to further improve the results seen in this approach. Furthermore, these additional factors might be useful when considering the changes to the Android permission system.

Finally, Android permissions are shown to be useful in the classification of malware in this paper. Further study could be done to determine if the same process can be applied to the app's "category". App stores and forums where users can download apps all categorize apps based on their function (*e.g.*, Games, Finance, Entertainment, *etc.*). If it is possible to classify the apps based on the permissions they define, it should be possible to construct a model of what an app for a particular category should look like. Creating a definition of what an app belonging to a category typically looks like could provide the ability for detecting anomalous apps that may be malicious in nature. These apps could be studied more closely than others to determine if they are indeed malware.

7. Conclusion

This paper describes a technique that supports binary and familial classification of malware. The technique im-

proves on previous results in terms of precision, recall, f-score while maintaining similar accuracy scores. Familial classification is done on a number of different and similar malware families. Also, the process shows the ability to distinguish between overprivileged applications and malware. To validate the results, three separate AV engine taxonomies are used in familial classification.

8. Acknowledgments

This work is supported by a fellowship from the Isaac L. Auerbach Cybersecurity Institute at Drexel University.

References

[1] Smartphone OS Market Share, Q1 2015. http://www.idc.com/prodserv/smartphone-os-market-share.jsp. Accessed: 2015-06-24.

[2] System Permissions - Android API Guide. https://developer.android.com/guide/topics/security/permissions.html. Accessed: 2015-06-24.

[3] The Volume of New Mobile Malware Tripled in 2015. http://www.kaspersky.com/about/news/virus/2016/The_Volume_of_New_Mobile_Malware_Tripled_in_2015. Accessed: 2016-04-01.

[4] Z. Aung and W. Zaw. Permission-based android malware detection. *International Journal of Scientific and Technology Research*, 2(3):228–234, 2013.

[5] V. Avdiienko, K. Kuznetsov, A. Gorla, A. Zeller, S. Arzt, S. Rasthofer, and E. Bodden. Mining apps for abnormal usage of sensitive data. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 426–436. IEEE, 2015.

[6] C. M. Bishop. *Pattern Recognition and Machine Learning*. springer, 2006.

[7] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[8] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani. Crowdroid: behavior-based malware detection system for android. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 15–26. ACM, 2011.

[9] S. Chakradeo, B. Reaves, P. Traynor, and W. Enck. Mast: triage for market-scale mobile malware analysis. In *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks*, pages 13–24. ACM, 2013.

[10] L. Deshotels, V. Notani, and A. Lakhotia. Droidlegacy: Automated familial classification of android malware. In *Proceedings of ACM SIGPLAN on Program Protection and Reverse Engineering Workshop 2014*, page 3. ACM, 2014.

[11] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, and M. Rajarajan. Android security: a survey of issues, malware penetration, and defenses. *Communications Surveys & Tutorials, IEEE*, 17(2):998–1022, 2015.

[12] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[13] L. Gomez and I. Neamtiu. A characterization of malicious android applications. Technical report, Technical report, University of California, Riverside, 2011.

[14] Google. PackageInfo — Android Developers. https://developer.android.com/reference/android/content/pm/PackageInfo.html. Accessed: 2015-12-17.

[15] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller. Checking app behavior against app descriptions. In *Proceedings of the 36th International Conference on Software Engineering*, pages 1025–1035. ACM, 2014.

[16] F. Idrees and M. Rajarajan. Investigating the android intents and permissions for malware detection. In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2014 IEEE 10th International Conference on*, pages 354–358. IEEE, 2014.

[17] B. Kang, B. Kang, J. Kim, and E. G. Im. Android malware classification method: Dalvik bytecode frequency analysis. In *Proceedings of the 2013 Research in Adaptive and Convergent Systems*, pages 349–350. ACM, 2013.

[18] F.-S. Labs. Threat Report: H2 2013. https://www.f-secure.com/documents/996508/1030743/Threat_Report_H2_2013.pdf. Accessed: 2015-06-24.

[19] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie. Whyper: Towards automating risk assessment of mobile applications. In *USENIX Security*, volume 13, 2013.

[20] U. Pehlivan, N. Baltaci, C. Acarturk, and N. Baykal. The analysis of feature selection methods and classification algorithms in permission based android malware detection. In *Computational Intelligence in Cyber Security (CICS), 2014 IEEE Symposium on*, pages 1–8. IEEE, 2014.

[21] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. G. Bringas, and G. Álvarez. Puma: Permission usage to detect malware in android. In *International Joint Conference CISIS12-ICEUTE´ 12-SOCO´ 12 Special Sessions*, pages 289–298. Springer, 2013.

[22] K. A. Talha, D. I. Alper, and C. Aydin. Apk auditor: Permission-based android malware detection system. *Digital Investigation*, 13:1–14, 2015.

[23] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu. Droidmat: Android malware detection through manifest and api calls tracing. In *Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on*, pages 62–69. IEEE, 2012.

[24] L.-K. Yan and H. Yin. Droidscope: Seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis. In *USENIX security symposium*, pages 569–584, 2012.

[25] 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 *Computer Security-ESORICS 2014*, pages 163–182. Springer, 2014.

[26] Y. Zhou. Android malware: Detection, characterization, and mitigation. 2015.

[27] Y. Zhou and X. Jiang. Dissecting android malware: Characterization and evolution. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 95–109. IEEE, 2012.